Міністерство освіти і науки України

Кравець В.О., Хавіна І.П., Колибін Ю.М.,
Нікітіна Л.О., Зиков І.С., Філоненко А.М., Хавін В.Л.

ВСТУП ДО ЕКСПЕРТНИХ СИСТЕМ

Рекомендовано Міністерством освіти і науки України
як навчальний посібник

НТУ “ХПІ” 2006

ББК 32.813
Х 11
УДК 004.891

Рецензенти: *В.П. Полтавцев*, канд. техн. наук, доц., нач. служби з інформац. технологій та зв'язку КП "Харківські теплові мережі";
О.Н. Фоменко, д-р техн. наук, проф., ХВУ

Автори: Кравець В.О., Хавіна І.П., Колибін Ю.М., Нікітіна Л.О., Зиков І.С., Філоненко А.М., Хавін В.Л.

Гриф надано Міністерством освіти і науки України,
лист № 14/18.2 –1094 від 16.05.2005 р.

Х11 Вступ до експертних систем: Навч. посіб. / Кравець В.О., Хавіна І.П. та ін. — Харків: НТУ "ХПІ", 2006. — 232 с.

ISBN _____

Навчальний посібник включає основні зведення по створенню експертних систем або систем, заснованих на знаннях. Посібник на прикладах навчальних задач дає уявлення про різноманітні підходи до побудови експертних систем.

Призначено для фахівців в області інформатики і штучного інтелекту, студентів і аспірантів вузів.

The manual includes the basic data on creation of expert systems or the systems based on knowledge. The grant to examples of educational problems gives representation about various approaches to construction of expert systems.

It is intended for experts in the field of computer science and an artificial intellect, students and post-graduate students of universities.

Іл. 39 Табл. 14 Бібліогр. 73 назв.

ББК 32.813

ISBN_____

Зміст

Передмова

У книзі зроблена спроба систематизувати знання і досвід, накопичені в області експертних систем, і представити їх у виді стиснутого курсу по методології експертних систем і їхнього застосування в реальних умовах.

Розділ 1 містить огляд стану ринку систем штучного інтелекту, структуру експертної системи (ЕС), етапи розробки ЕС і представлення знань.

Розділ 2 присвячений розгляданню двох систем логіки - вирахуванню висловлень і вирахуванню предикатів. Описано формальну систему доказу формул вирахування висловлень і предикатів.

У главі 3 описані методи представлення знань предметної області – семантичні мережі, фрейми, правила продукцій.

Розділ 4 присвячений методам рішення задач і приведені програми з використанням евристичних оцінних функцій.

Глави 5 і 6 знайомлять з деякими принципами організації ЕС і імовірнісними міркуваннями при пошуку рішень в умовах невизначеності і висновку в мережі при наявності конкуруючих гіпотез.

Комбіноване використання експертної системи й апарата штучних нейронних мереж, описаних у главі 7, забезпечує необхідну гнучкість і самонавчання на основі знань.

Опис оболонки G2- графічної, об'єктно – орієнтованого середовища для побудови і супроводу експертних систем реального часу, призначених для моніторингу, діагностики, оптимізації, планування і керування динамічними процесами, приведено в главі 8.

Інформаційні ресурси Інтернет, використовувані при підготовці даного посібника перераховані в главі 9.

1. Введення в експертні системи

На початку вісімдесятих років у дослідженнях зі штучного інтелекту сформувався самостійний напрямок, що одержав назву "експертні системи" (ЕС). Ціль досліджень по ЕС складається в розробці програм, що при рішенні задач, важких для експерта-людини, одержують результати, що не уступають по якості й ефективності рішенням, одержуваним експертом. Дослідники в області ЕС для назви своєї дисципліни часто використовують також термін "інженерія знань", введений Е. Фейгенбаумом (Feigenbaum) [4] як "привнесення принципів і інструментарію досліджень з області штучного інтелекту в рішення важких прикладних проблем, що вимагають знань експертів".

Програмні засоби (ПЗ), що базуються на технології експертних систем, або інженерії знань (надалі будемо використовувати їх як синоніми), одержали значне поширення у світі. Важливість експертних систем полягає в наступному:

- технологія експертних систем істотно розширює коло практично значимих задач, розв'язуваних на комп'ютерах, рішення яких приносить значний економічний ефект;
- технологія ЕС є найважливішим засобом у рішенні глобальних проблем традиційного програмування: тривалість і, отже, висока вартість розробки складних додатків; висока вартість супроводу складних систем, що часто в кілька разів перевершує вартість їхньої розробки; низький рівень повторної використовуваності програм і т.п.;
- об'єднання технології ЕС з технологією традиційного програмування додає нові якості до програмних продуктів за рахунок забезпечення динамічної модифікації додатків користувачем, а не програмістом; більшої "прозорості" додатка (наприклад, знання зберігаються на обмеженій природній мові (ПМ), що не вимагає коментарів до знань, спрощує навчання і супровід); кращої графіки; інтерфейсу і взаємодії.

На думку провідних спеціалістів [8], у недалекій перспективі ЕС знайдуть наступне застосування:

- ЕС будуть відігравати ведучу роль у всіх фазах проектування, розробки, виробництва, розподілу, продажу, підтримки і надання послуг;
- технологія ЕС, що одержала комерційне поширення, забезпечить революційний прорив в інтеграції додатків з готових інтелектуально - взаємодіючих модулів.

ЕС призначені для так званих неформалізованих задач, тобто ЕС не відкидають і не заміняють традиційного підходу до розробки програм, орієнтованого на рішення формалізованих задач, що володіють однією або декількома з наступних характеристик:

- задачі не можуть бути задані в числовій формі;
- цілі не можуть бути виражені в термінах точно визначеної цільової функції;
- не існує алгоритмічного рішення задач;
- Неформалізовані задачі звичайно володіють:
- помилковістю, неоднозначністю, неповнотою і суперечливістю вихідних даних;
- помилковістю, неоднозначністю, неповнотою і суперечливістю знань про проблемну область і розв'язувану задачу;
- великою розмірністю простору рішення, тобто перебір при пошуку рішення досить великий;
- динамічно змінюються даними і знаннями.

Варто підкреслити, що неформалізовані задачі представляють великий і дуже важливий клас задач.

Експертні системи і системи штучного інтелекту відрізняються від систем обробки даних тим, що в них в основному використовуються символьний (а не числовий) спосіб представлення, символьний висновок і евристичний пошук рішення (а не виконання відомого алгоритму).

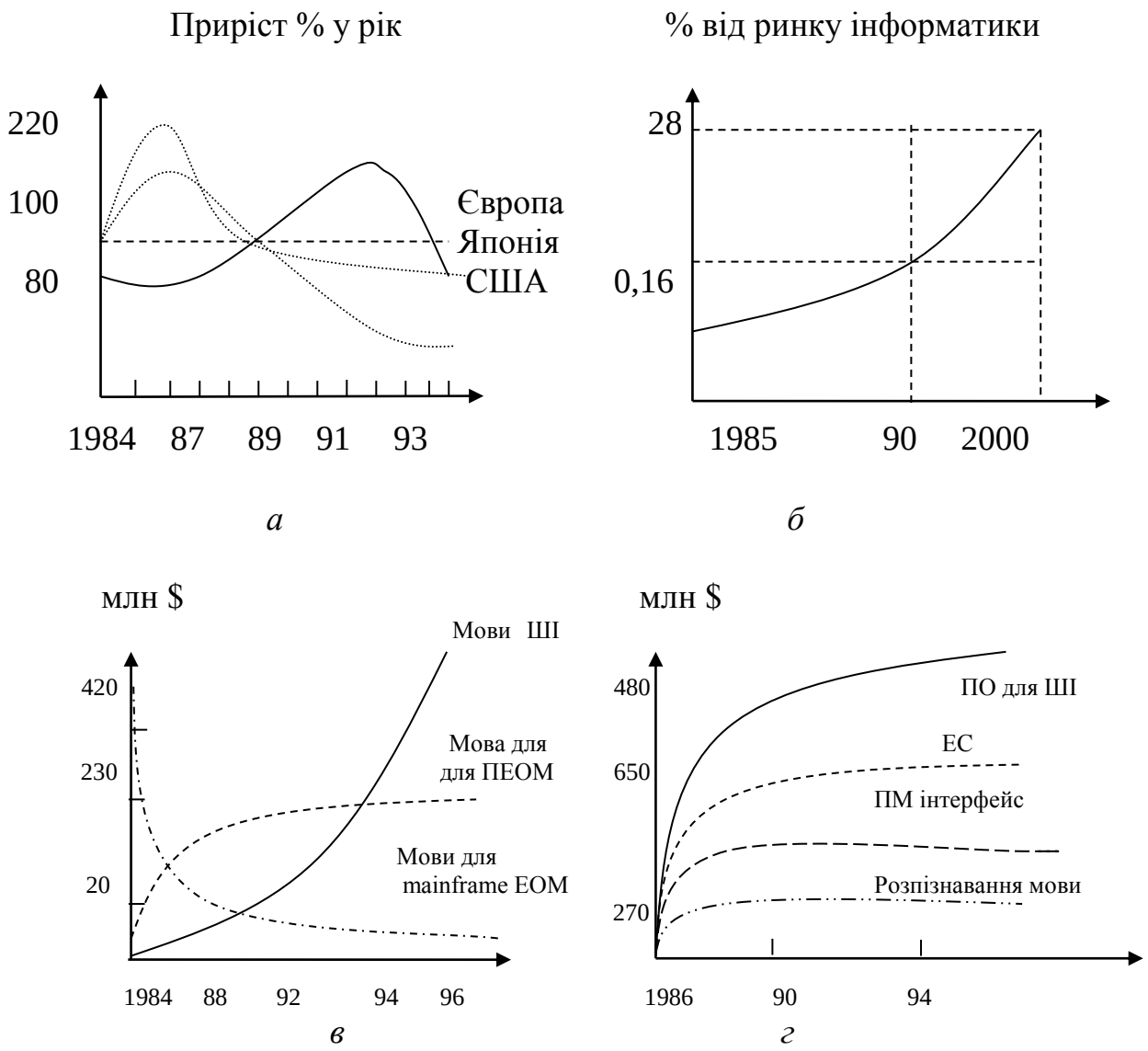
По якості й ефективності рішення експертні системи не уступають рішенням експерта-людини. Рішення експертних систем володіють *"прозорістю",* тобто можуть бути пояснені користувачеві на якісному рівні. Ця якість експертних систем забезпечується їхньою здатністю міркувати про свої знання й умовиводи. Експертні системи здатні поповнювати свої знання в ході взаємодії з експертом.

Технологія експертних систем використовується для рішення різних типів задач: інтерпретація, пророкування, діагностика, планування, конструювання, контроль, налагодження, інструктаж, керування. Застосовується в найрізноманітніших проблемних областях, таких, як фінанси, нафтова і газова промисловість, енергетика, транспорт, фармацевтичне виробництво, космос, металургія, гірська справа, хімія, утворення, целюлозно-паперова промисловість, телекомунікації і зв'язок і ін.

Комерційні успіхи до фірм-розроблювачів систем штучного інтелекту (СШІ) прийшли не відразу. Протягом 1960 - 1985 р. успіхи ШІ стосувалися в основному дослідницьких розробок, що демонстрували придатність СШІ для практичного використання. Починаючи приблизно з 1985 р. (у масовому масштабі з 1988 - 1990 р.), у першу чергу ЕС, а в останні роки системи, що сприймають природну мову (ПМ-системи), і нейроні мережі (НМ) стали активно використовуватися в комерційних додатках.

Малюнок 1.1. відбиває [65] різні аспекти стану ринку ШІ: інвестиції в розробку в області ШІ (США, Європа, Японія) (Малюнок 1.1., *а)*; частка систем ШІ в інформатиці (програмному забезпеченні) (Малюнок 1.1., *б)*; доходи від продажів традиційних мов програмування (Малюнок 1.1., *в)*; інвестиції тільки в програмне забезпечення (США) (Малюнок 1.1., *г)*.

Треба відзначити, що на зорі появи ЕС специфіка використовуваних у них мов, технології розробки додатків і використовуваного спеціального устаткування (наприклад, Lisp-машини) давала підстави припускати, що інтеграція ЕС із традиційними, програмними системами є складною і, можливо, нездійсненною задачею при обмеженнях, що накладаються реальними додатками.



Малюнок 1.1. Стан і перспективи ринку ШІ

Однак у даний час комерційні інструментальні засоби (ІЗ) для створення ЕС розробляються в повній відповідності із сучасними технологічними тенденціями традиційного програмування, що знімає проблеми, що виникають при створенні інтегрованих додатків.

Причини, що привели СШІ до комерційного успіху, наступні [57].

Інтегрованість. Розроблено інструментальні засоби штучного інтелекту (ІЗ ШІ), що легко інтегруються з іншими інформаційними технологіями і засобами (з CASE, СКБД, контролерами, концентраторами даних і т.п.).

Відкритість і переносність. ІЗ ШІ розробляються з дотриманням стандартів, що забезпечують відкритість і переносність.

Використання мов традиційного програмування і робочих станцій. Перехід від ІЗ ШІ , реалізованих на мовах ШІ (Lisp, Prolog і т.п.), до ІЗ ШІ , реалізованим на мовах традиційного програмування (С, С++ і т.п.), спростив забезпечення інтегрованості, знизив вимоги додатків ШІ до швидкодії ЕОМ і обсягам оперативної пам'яті. Використання робочих станцій (замість ПК) різко збільшило коло додатків, що можуть бути виконані на ЕОМ з використанням ІЗ ШІ .

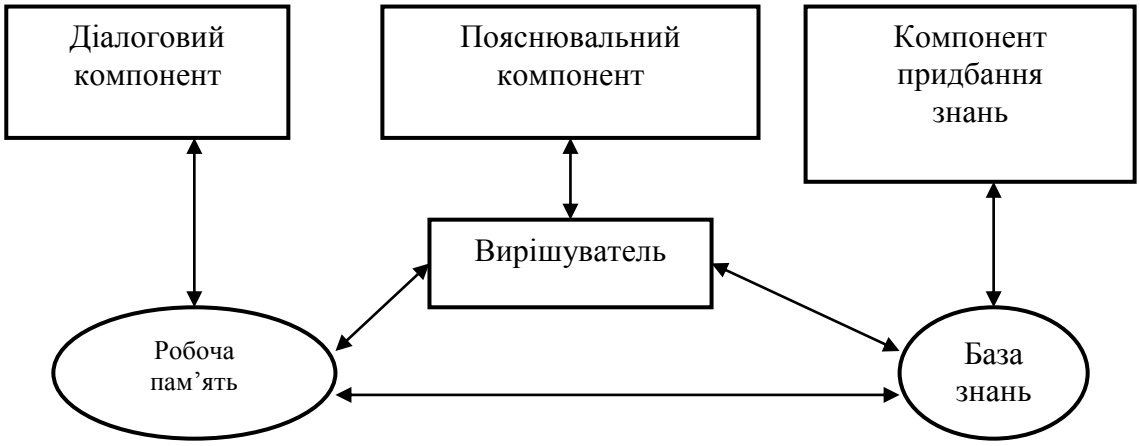
Архітектура клієнт-сервер. Розроблені ІЗ ШІ , що підтримують розподілені обчислення по архітектурі клієнт-сервер, що дозволило: знизити вартість устаткування, використовуваного в додатках, децентралізувати додатки, підвищити надійність і загальну продуктивність (тому що скорочується кількість інформації, що пересилається між ЕОМ, і кожен модуль додатка виконується на адекватному йому устаткуванні).

Проблемно / предметно - орієнтовані ІЗ ШІ . Перехід від розробок ІЗ ШІ загального призначення (хоча вони не утратили своє значення як засіб для створення орієнтованих ІЗ) до проблемно / предметно - орієнтованим ІЗ ШІ [11] забезпечує: скорочення термінів розробки додатків; збільшення ефективності використання ІЗ; спрощення і прискорення роботи експерта; повторне використання інформаційного і програмного забезпечення (об'єкти, класи, правила, процедури).

1.1.Структура експертних систем

ЕС складається [57] з наступних основних компонентів: вирішувателя (інтерпретатора); робочої пам'яті (РП), називаною також базою даних (БД); бази знань (БЗ); компонентів придбання знань; пояснювального компонента; діалогового компонента.

База даних (робоча пам'ять) призначена для збереження вихідний і проміжний даних розв'язуваної в сучасний момент задачі. Цей термін збігається за назвою, але не за змістом з терміном, використовуваним в інформаційно-пошукових системах (ІПС) і системах керування базами даних (СКБД) для позначення всіх даних (у першу чергу довгострокових), збережених у системі.



Малюнок 1.2 Основні компоненти ЄС

База знань (БЗ) у ЕС призначена для збереження довгострокових даних, що описують розглянуту область (а не поточних даних), і правил, що описують доцільні перетворення даних цієї області.

Вирішуватель, використовуючи вихідні дані з робочої пам'яті і знання з БЗ, формує таку послідовність правил, що приводить до рішення задачі.

Компонент придбання знань автоматизує процес наповнення ЕС знаннями, здійснюваний користувачем-експертом.

Пояснювальний компонент пояснює, як система одержала рішення задачі (або чому вона не одержала рішення) і які знання вона при цьому використовувала, що полегшує експертові тестування системи і підвищує довіру користувача до отриманого результату.

Діалоговий компонент орієнтований на організацію дружнього спілкування з користувачем як у ході рішення задач, так і в процесі придбання знань і пояснення результатів роботи.

У розробці ЕС беруть участь представники наступних спеціальностей:

експерт проблемної області, задачі якої буде вирішувати ЕС;

інженер по знаннях - фахівець з розробки ЕС (використовувану їм технологію, методи називають технологією (методами) інженерії знань);

програміст по розробці інструментальних засобів (ІЗ), призначених для прискорення розробки ЕС.

Необхідно відзначити, що відсутність серед учасників розробки інженерів по знаннях (тобто їхня заміна програмістами) або приводить до невдачі процес створення ЕС, або значно подовжує його.

Експерт визначає знання (дані і правила), що характеризують проблемну область, забезпечує повноту і правильність введених у ЕС знань.

Інженер по знаннях допомагає експертові виявити і структурувати знання, необхідні для роботи ЕС; здійснює вибір того ІЗ, що найбільше підходить для даної проблемної області, і визначає спосіб представлення знань у цьому ІЗ; виділяє і програмує (традиційними засобами) стандартні функції (типові для даної проблемної області), що будуть використовуватися в правилах, що вводяться експертом.

Програміст розробляє ІЗ (якщо ІЗ розробляється заново), що містить у межі всі основні компоненти ЕС, і здійснює його сполучення з тим середовищем, у якій воно буде використано.

Експертна система працює в двох режимах: режимі придбання знань і в режимі рішення задачі (називаному також режимом консультації або режимом використання ЕС).

У режимі придбання знань спілкування з ЕС здійснює (за посередництвом інженера по знаннях) експерт. У цьому режимі експерт, використовуючи компонент придбання знань, наповняє систему знаннями, що дозволяють ЕС у режимі рішення самостійно (без експерта) вирішувати задачі з проблемної

області. Експерт описує проблемну область у виді сукупності даних і правил. Дані визначають об'єкти, їхні характеристики і значення, що існують в області експертизи. Правила визначають способи маніпулювання з даними, характерні для розглянутої області.

Відзначимо, що режимові придбання знань у традиційному підході до розробки програм відповідають етапи алгоритмізації, програмування і налагодження, виконувані програмістом. Таким чином, на відміну від традиційного підходу у випадку ЕС розробку програм здійснює не програміст, а експерт (за допомогою ЕС), що не володіє програмуванням.

У режимі консультації спілкування з ЕС здійснює кінцевий користувач, якого цікавить результат і (або) спосіб його одержання. Необхідно відзначити, що в залежності від призначення ЕС користувач може не бути фахівцем у даній проблемній області (у цьому випадку він звертається до ЕС за результатом, не вміючи одержати його сам), або бути фахівцем (у цьому випадку користувач може сам одержати результат, але він звертається до ЕС з метою або прискорити процес одержання результату, або покласти на ЕС рутинну роботу). У режимі консультації дані про задачу користувача після обробки їхнім діалоговим компонентом надходять у робочу пам'ять. Вирішуватель на основі вхідних даних з робочої пам'яті, загальних даних про проблемну область і правил із БЗ формує рішення задачі. ЕС при рішенні задачі не тільки виконує запропоновану послідовність операції, але і попередньо формує її. Якщо реакція системи не зрозуміла користувачеві, то він може зажадати пояснення:

"Чому система задає те або інше питання?", "як відповідь, що збирається системою, отриманий?".

Структуру (Малюнок 1.3) називають **структурою статичної ЕС**. ЕС даного типу використовуються в тих додатках, де можна не враховувати зміни навколишнього світу, що відбуваються за час рішення задачі. Перші ЕС, що одержали практичне використання, були статичними.

В архітектуру динамічної ЕС у порівнянні зі статичної ЕС уводяться два компоненти: підсистема моделювання зовнішнього світу і підсистема зв'язку з

зовнішнім оточенням. Остання здійснює зв'язок з зовнішнім світом через систему датчиків і контролерів. Крім того, традиційні компоненти статичної ЕС (база знань і машина висновку) перетерплюють істотні зміни, щоб відбити тимчасову логіку подій, що відбуваються в реальному світі.

1.2. Класифікація систем, заснованих на знаннях

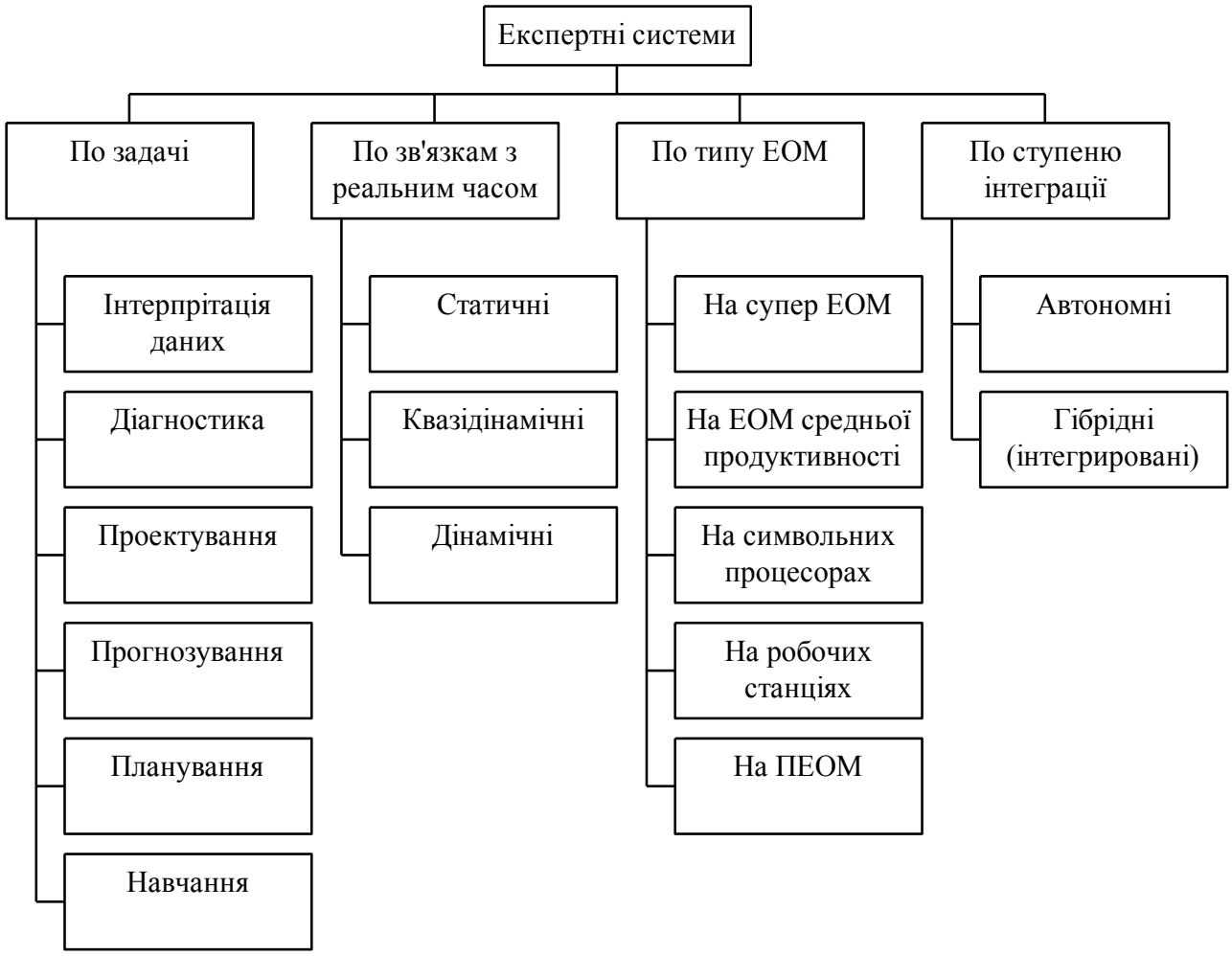
Клас ЕС сьогодні поєднує кілька тисяч різних програмних комплексів, які можна класифікувати за різними критеріями [65]. Одна з можливих класифікацій (Малюнок 1.3).

1.3.Інтерпретація даних

Це одна з традиційних задач для ЕС. Інтерпретація – процес визначення змісту даних, результати якого повинні бути погодженими і коректними. Звичайно передбачається різноманітний аналіз даних.

Приклади:

- Виявлення й ідентифікація різних типів океанських судів за результатами аерокосмічного сканування – SIAP.
- Визначення основних типів властивостей особистості за результатами психодіагностичного тестування в системах АВАНТЕСТ і МІКРОЛЮШЕР.



Малюнок 1.3 Класифікація експертних систем

1.4.Діагностика

Діагностика – процес співвіднесення об'єкта з деяким класом об'єктів і/або виявлення несправності в деякій системі. Несправність – це відхилення від норми. Таке трактування дозволяє з єдиних теоретичних позицій розглядати і несправність устаткування в технічних системах, і захворювання живих організмів, і всілякі природні аномалії. Важливою специфікою є тут необхідність розуміння функціональної структури («анатомії») діагностуючої системи.

Приклади:

- Діагностика і терапія звуження коронарних судин – ANGY.
- Діагностика помилок в апаратурі і математичному забезпеченні ЕОМ – CRIB.

1.5. Моніторинг

Основна задача моніторингу – безперервна інтерпретація даних у реальному масштабі часу і сигналізація про вихід тих або інших параметрів за припустимі межі. Головні проблеми – пропуск тривожної ситуації й інверсна задача «помилкового» спрацьовування. Складність цих проблем у розмитості симптомів тривожних ситуацій і необхідність обліку тимчасового контексту.

Приклади:

- Контроль роботи електростанцій СПРИНТ, допомога диспетчерам атомного реактора – REACTOR.
- Контроль аварійних датчиків на хімічному заводі – FALCON.

1.6.Проектування

Проектування складається в підготовці специфікацій на створення «об'єктів» із заздалегідь визначеними властивостями. Під специфікацією розуміється весь набір необхідних документів – креслення, пояснювальна записка і т.д. Основні проблеми – одержання чіткого структурованого опису знань про об'єкт і проблема «сліду». Для організації ефективного проектування й у ще більшому ступені перепроектування необхідно формувати не тільки самі проектні рішення, але і мотиви їхнього прийняття. Таким чином, у задачах проектування тісно зв'язуються два основних процеси. Виконуваних у рамках відповідної ЕС: процес висновку рішення і процес пояснення.

Приклади:

- Проектування конфігурацій ЕОМ VAX-11/780 у системі XCON, проектування БІС – CADHELP.
- Синтез електричних ланцюгів – SYN.

1.7.Прогнозування

Прогнозування дозволяє пророчити наслідок деяких подій або явищ на підставі аналізу наявних даних. Прогнозуючі системи логічно виводять ймовірні наслідки з заданих ситуацій. У прогнозуючій системі звичайно використовується параметрична динамічна модель, у якій значення параметрів «підганяються» під задану ситуацію. Виведені з цієї моделі висновки складають основу для прогнозів з імовірнісними оцінками.

Приклади:

- Пророкування погоди – WIILARD.
- Оцінки майбутнього врожаю – PLANT.
- Прогнози в економіці – ECON.

1.8.Планування

Під плануванням розуміється знаходження планів дій, що відносяться до об'єктів, здатних виконувати деякі функції. У таких ЕС використовуються моделі поводження реальних об'єктів для того, щоб логічно вивести наслідок планованої діяльності. Приклади:

- Планування поводження роботи – STRIPS.
- Планування промислових замовлень – ISIS.
- Планування експерименту – MOLGEN.

1.9.Навчання

Під навчанням розуміється використання комп'ютера для навчання якійсь дисципліні або предметові. Системи навчання діагностують помилки при вивченні якої-небудь дисципліни за допомогою комп'ютера і підказують правильні рішення. Вони містять знання про гіпотетичного «учня» і його характерних помилках, потім у роботі вони здатні діагностувати слабкі місця в пізнаннях тих, яких навчають, і знаходити відповідні засоби для їхньої ліквідації.

Крім того, вони планують акт спілкування з учнем у залежності від успіхів учня з метою передачі знань.

Приклади:

- Навчання мові програмування LISP у системі «учитель Ліспа».
- Навчання мові Паскаль – система PROUST.

1.10. Керування

Під керуванням розуміється функція організованої системи, що підтримує визначений режим діяльності. Такого роду ЕС здійснюють керування поводженням складних систем із заданими специфікаціями.

Приклади:

- Допомога в керуванні газової котельні – GAS.
- Керування системою календарного планування – Project Assistant.

1.11. Підтримка прийняття рішень

Підтримка прийняття рішень – це сукупність процедур, що забезпечує особу, що приймає рішення, необхідною інформацією і рекомендаціями, що полегшують процес ухвалення рішення. Ці ЕС допомагають фахівцям вибрати і/або сформувати потрібну альтернативу серед безлічі виборів при прийнятті рішень.

Приклади:

- Вибір стратегії виходу фірми з кризової ситуації – CRYSIS.
- Допомога у виборі страхової компанії або інвестора – CHOICE.

У загальному випадку, усі системи, засновані на знаннях, можна підрозділити на *системи, що вирішують задачі аналізу і системи, що вирішують задачі синтезу*. Основна відмінність задач аналізу від задач синтезу полягає в тім, що якщо в задачах аналізу безліч рішень може бути перерахована і включена в систему, то в задачах синтезу безліч рішень потенційно не обмежена і будується з компонентів або підпроблем. Задачами аналізу є інтерпретація даних, діагностика,

підтримка ухвалення рішення; до задач синтезу відноситься проектування, планування, керування. Комбіновані задачі – навчання, моніторинг, прогнозування.

1.12. Класифікація по зв'язку з реальним часом

Статичні ЕС розробляються в предметних областях, у яких база знань і інтерпретуючі дані не міняються в часі. Вони стабільні.

Приклад:

- Діагностика несправностей в автомобілі.

Квазідинамічні ЕС інтерпретують ситуацію, що міняється з деяким фіксованим інтервалом часу.

Приклад:

- Мікробіологічні ЕС, у яких знімаються лабораторні виміри з технологічного процесу один раз у 4-5 годин (наприклад, виробництво лізина) і аналізується динаміка отриманих показників стосовно попереднього виміру.

Динамічні ЕС працюють у сполученні з датчиками об'єктів у режимі реального часу з безперервною інтерпретацією даних, що надходять у систему.

Приклади:

- Керування гнучкими виробничими комплексами, моніторинг у реанімаційних палатах.
- Програмний інструментарій для розробки динамічних систем – G2.

1.13. Класифікація по типам ЕОМ

На сьогоднішній день існують:

- ЕС для унікальних стратегічно важливих задач на супер-еом – CRAY, CONVEX.
- ЕС на ЕОМ середній продуктивності типу Mainframe.
- ЕС на символьних процесорах і робочих станціях SUN, Silicon Graphics, APOLLO.
- ЕС на міні- і суперміні-еом VAX, micro-VAX.
- ЕС на персональних комп'ютерах.

1.14. Класифікація по ступені інтеграції з іншими програмами

Автономні ЕС працюють безпосередньо в режимі консультацій з користувачем для специфічних «експертних» задач, для рішення яких не потрібно залучати традиційні методи обробки даних (розрахунки, моделювання і т.д.).

Гібридні ЕС представляють програмний комплекс, агрегуючий стандартні пакети прикладних програм (ППП) (наприклад, матстатистика, лінійне програмування, СКБД) і засобу маніпулювання знаннями. Це може бути інтелектуальна надбудова над ППП або інтегроване середовище для рішення складної задачі з елементами експертних знань.

Незважаючи на зовнішню привабливість гібридного підходу, слід зазначити, що розробка таких систем являє собою задачу, на порядок більш складну, чим розробка автономної ЕС. Стикування не просто різних пакетів, а різних методологій (що відбувається в гібридних системах) породжують цілий комплекс теоретичних і практичних труднощів.

1.15. Етапи розробки експертних систем

Розробка ЕС має істотні відмінності від розробки звичайного програмного продукту. Досвід створення ЕС показав, що використання при їхній розробці методології, прийнятої в традиційному програмуванні, або надмірно затягує процес створення ЕС, або взагалі приводить до негативного результату.

Використовувати ЕС слід тільки тоді, коли розробка ЕС *можлива, виправдана і* методи інженерії знань *відповідають* розв'язуваній задачі. Щоб розробка ЕС була *можливою* для даного додатка, необхідне одночасне виконання принаймні наступних вимог:

1) існують експерти в даній області, що вирішують задачу значно краще, ніж починаючі фахівці;

2) експерти сходяться в оцінці пропонованого рішення, інакше не можна буде оцінити якість розробленої ЕС;

3) експерти здатні вербалізувати (виразити природною мовою) і пояснити використовувані ними методи, у противному випадку важко розраховувати на те, що знання експертів будуть "витягнуті" і вкладені в ЕС;

4) рішення задачі вимагає тільки міркувань, а не дій;

5) задача не повинна бути занадто важкої (тобто її рішення повинне займати в експерта кілька годин або днів, а не тижнів);

6) задача хоча і не повинна бути виражена у формальному виді, але все-таки повинна відноситися до досить "зрозумілої" і структурованої області, тобто повинні бути виділені основні поняття, відносини і відомі (хоча б експертові) способи одержання рішення задачі;

7) рішення задачі не повинне в значній мірі використовувати "здоровий глузд" (тобто широкий спектр загальних зведень про світ і про спосіб його функціонування, що знає і уміє використовувати будь-яка нормальна людина), тому що подібні знання поки не вдається (у достатній кількості) вкласти в системи штучного інтелекту.

Використання ЕС у даному додатку може бути можливо, але не виправдано. Застосування ЕС може бути *виправдано* одним з наступних факторів:

- рішення задачі принесе значний ефект, наприклад економічний;
- використання людини-експерта неможливо або через недостатню кількість експертів, або через необхідність виконувати експертизу одночасно в різних місцях;
- використання ЕС доцільно в тих випадках, коли при передачі інформації експертові відбувається неприпустима втрата часу або інформації;

Додаток *відповідає* методам ЕС, якщо розв'язувана задача має сукупність наступних характеристик:

1) задача може бути природним образом вирішена за допомогою маніпуляції із символами (тобто за допомогою символічних міркувань), а не маніпуляцій з числами, як прийнято в математичних методах і в традиційному програмуванні;

2) задача повинна мати евристичну, а не алгоритмічну природу, тобто її рішення повинне вимагати застосування евристичних правил. Задачі, що можуть бути гарантовано вирішені (з дотриманням заданих обмежень) за допомогою деяких формальних процедур, не підходять для застосування ЕС;

3) задача повинна бути досить складна, щоб виправдати витрати на розробку ЕС. Однак вона не повинна бути надмірно складною (рішення займає в експерта години, а не тижні), щоб ЕС могла нею вирішувати;

4) задача повинна бути досить вузькою, щоб вирішуватися методами ЕС, і практично значимою.

При розробці ЕС, як правило, використовується концепція "швидкого прототипу". Суть цієї концепції полягає в тому, що розроблювачі не намагаються відразу побудувати кінцевий продукт. На початковому етапі вони створюють прототип (прототипи) ЕС. Прототипи повинні задовольняти двом суперечливим вимогам: з одного боку, вони повинні вирішувати типові задачі конкретного додатка, а з іншого боку - час і трудомісткість їхньої розробки повинні бути досить незначні, щоб можна було максимально запаралелити процес нагромадження і налагодження знань (здійснюваний експертом) із процесом

вибору програмних засобів. Для задоволення зазначеним вимогам, як правило, при створенні прототипу використовуються різноманітні засоби, що прискорюють процес проектування.

Прототип повинний продемонструвати придатність методів інженерії знань для даного додатка. У випадку успіху експерт за допомогою інженера по знаннях розширює знання прототипу про проблемну область. При невдачі може знадобитися розробка нового прототипу або розроблювачі можуть прийти до висновку про непридатність методів ЕС для даного додатка. В міру збільшення знань прототип може досягти такого стану, коли він успішно вирішує всі задачі даного додатка. Перетворення прототипу ЕС у кінцевий продукт звичайно приводить до перепрограмування ЕС на мовах низького рівня, що забезпечують як збільшення швидкодії ЕС, так і зменшення необхідної пам'яті. Трудомісткість і час створення ЕС у значній мірі залежать від типу використовуваного інструментарію.

У ході робіт зі створення ЕС склалася визначена технологія їхньої розробки, що включає шість наступних етапів (Малюнок 1.4):ідентифікацію, концептуалізацію, формалізацію, виконання, тестування, досвідчену експлуатацію. На етапі *ідентифікації* визначаються задачі, що підлягають рішенню, виявляються цілі розробки, визначаються експерти і типи користувачів.

На етапі *концептуалізації* проводиться змістовний аналіз проблемної області, виявляються використовувані поняття і їхні взаємозв'язки, визначаються методи рішення задач.

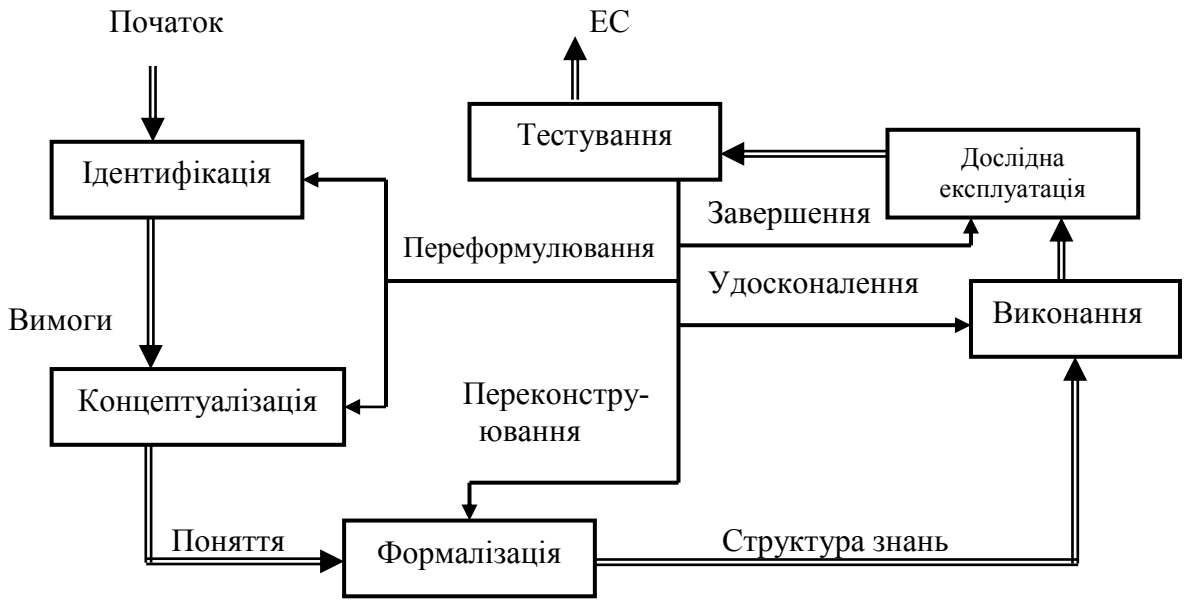
На етапі *формалізації* вибираються ІЗ і визначаються способи представлення усіх видів знань, формалізуються основні поняття, визначаються способи інтерпретації знань, моделюється робота системи, оцінюється адекватність цілям системи зафіксованих понять, методів рішень, засобів представлення і маніпулювання знаннями.

На етапі *виконання* здійснюється наповнення бази знань, створення прототипу ЕС. Головне в створенні прототипу полягає в тім, щоб цей прототип

забезпечив перевірку адекватності ідей, методів і способів представлення знань розв'язуваним задачам.

Створення першого прототипу повинне підтвердити, що обрані методи рішень і способи представлення придатні для успішного рішення, принаймні, ряду задач з актуальної предметної області, а також продемонструвати тенденцію до одержання високоякісних і ефективних рішень для всіх задач предметної області в міру збільшення обсягу знань.

У ході етапу *тестування* виробляється оцінка обраного способу представлення знань у ЕС у цілому. Для цього інженер по знаннях підбирає приклади, що забезпечують перевірку всіх можливостей нової ЕС.



Малюнок 1.4 Технологія розробки ЕС

Розрізняють наступні джерела невдач у роботі системи: тестові приклади, уведення-вивід, правила виводу, керуючі стратегії.

Показові тестові приклади є найбільш очевидною причиною невдалої роботи ЕС. Тому при підготовці тестових прикладів варто класифікувати їх по під

проблемам предметної області, виділяючи стандартні випадки, визначаючи границі важких ситуацій і т.п.

Критерії оцінки ЕС залежать від точки зору. При тестуванні промислової системи превалює точка зору інженера по знаннях, якого в першу чергу цікавить питання оптимізації представлення і маніпулювання знаннями. І, нарешті, при тестуванні ЕС після досвідченої експлуатації оцінка виробляється з погляду користувача, зацікавленого в зручності роботи й одержання практичної користі.

На етапі *досвідченої експлуатації* перевіряється придатність ЕС для кінцевого користувача. Придатність ЕС для користувача визначається в основному зручністю роботи з нею і її корисністю. Під корисністю ЕС розуміється її здатність у ході діалогу визначати потреби користувача, виявляти й усувати причини невдач у роботі, а також задовольняти зазначені потреби користувача (вирішувати поставлені задачі). У свою чергу, зручність роботи з ЕС має на увазі природність взаємодії з нею (спілкування в звичному, не стомлюючого користувача виді), гнучкість ЕС (здатність системи набудовуватися на різних користувачів, а також враховувати зміни в кваліфікації того самого користувача) і стійкість системи до помилок (здатність не виходити з ладу при помилкових діях недосвідченого користувачах).

У ході розробки ЕС майже завжди здійснюється її модифікація. Виділяють наступні види модифікації системи: переформулювання понять і вимог, переконструювання представлення знань у системі й удосконалення прототипу.

1.16. Представлення знань в експертних системах

Перше й основне питання, яке треба вирішити при представленні знань - це питання визначення складу знань, тобто визначення того, "ЩО ПРЕДСТАВЛЯТИ" в експертній системі. Друге запитання стосується того, "ЯК ПРЕДСТАВЛЯТИ" знання. Необхідно відзначити, що ці дві проблеми не є незалежними. Дійсно, обраний спосіб представлення може виявитися непридатним у принципі або неефективним для вираження деяких знань.

Питання "ЯК ПРЕДСТАВЛЯТИ" можна розділити на дві у значній мірі незалежні задачі: як організувати (структурувати) знання і як представити знання в обраному формалізмі.

Прагнення виділити організацію знань у самостійну задачу викликано, зокрема, тим, що ця задача виникає для будь-якої мови представлення і способи рішення цієї задачі є однаковими (або подібними) поза залежністю від використовуваного формалізму.

Отже, у коло питань, розв'язуваних при представленні знань, будемо включати наступні:

- визначення складу знань, що представляються;
- організацію знань;
- представлення знань, тобто визначення моделі представлення.

Склад знань ЕС визначається наступними факторами:

- проблемним середовищем;
- архітектурою експертної системи;
- потребами і цілями користувачів;
- мовою спілкування.

Відповідно до загальної схеми статичної експертної системи (див. мал.1.1) для її функціонування потрібні наступні знання:

- знання про процес рішення задачі (тобто керуючі знання), використовувані інтерпретатором (вирішувателем);
- знання про мову спілкування і способи організації діалогу, використовувані лінгвістичним процесором (діалоговим компонентом);
- знання про способи представлення і модифікації знань, використовувані компонентом придбання знань;
- підтримуючі структурні і керуючі знання, використовувані пояснювальним компонентом.

Для динамічної ЕС, крім того, необхідні наступні знання:

1) знання про методи взаємодії з зовнішнім оточенням;

2) знання про моделі зовнішнього світу.

Залежність складу знань від вимог користувача виявляється в наступному:

- які задачі (із загального набору задач) і з якими даними хоче вирішувати користувач;
- які кращі способи і методи рішення;
- при яких обмеженнях на кількість результатів і способи їхнього одержання повинна бути вирішена задача;
- які вимоги до мови спілкування й організації діалогу;
- яка ступінь спільності (конкретності) знань про проблемну область, доступна користувачеві;
- які цілі користувачів.

Склад знань про мову спілкування залежить як від мови спілкування, так і від необхідного рівня розуміння.

Контрольні питання

1. Для кожної з перерахованих програм вирішите, чи є вона ЕС
   - Програма прогнозування місцевої погоди.
   - Програма для виявлення несправності, якщо автомобіль не заводиться.
   - Програма для визначення стратегії гонки в режимі реального часу .
   - Програма для визначення оптимального маршруту комівояжера.
   - Програма допомоги по телефону довіри, коли необхідно визначити отруту, яку міг бути прийняти той що дзвонить.
   - Програма одержання тривимірного зображення будинку на підставі словесного опису пристрою і розмірів будинку.

2. Формальні основи експертних систем

2.1. Вирахування предикатів

Слово "логіка" означає систематичний метод міркувань. Розглянемо дві конкретні системи логіки - базисну (вирахування висловлень) і більш багату (вирахування предикатів). Вирахування висловлень - сукупність правил, використовуваних для визначення істинності або хибності деякої комбінації висловлень. У логіку розроблені добре визначені і зрозумілі формалізми для представлення фактів і правил.

Можна розглядати пропозиції або висловлення повсякденної розмовної мови, і за допомогою вирахування робити елементарні міркування.

Висловлення, або пропозиція, або факт - це просте твердження, що може бути істинним або ложним. Приклади пропозицій:

ЯКЩО *Сміт є фахівцем з ЕОМ,* ТО *Сміт – оптиміст.*

СМІТ є фахівцем з ЕОМ.

З цих правил можна зробити висновок:

Смит – оптиміст.

Більш формально виразимо це, застосувавши запис:

ЯКЩО P то Q, P ОТЖЕ Q

де P і Q попередні правила. Знайшовши збіг, людина може затверджувати, що судження має деяку прийнятну логічну структуру, і отже, може зробити висновок про те, що висновок

Сміт - -оптиміст

справедливо. Цілком твердження виглядає в такий спосіб:

ЯКЩО *Сміт є фахівцем з ЕОМ,* ТО *Сміт – оптиміст.*

СМІТ *є фахівцем з ЕОМ.*

ОТЖЕ

Сміт – оптиміст.

На обґрунтованості приклада ніяк не позначається, чи має зміст твердження яке-небудь значення у реальному світі. Наприклад, що випливає твердження справедливе, хоча й абсурдне:

ЯКЩО *Сміт є фахівцем з ЕОМ*, ТО *Сміт – спить.*

СМІТ *є фахівцем з ЕОМ.*

ОТЖЕ

Сміт – спить.

Факти можна переписати у виді предикатів:

є (сміт, фахівець з ЕОМ)

є (сміт, оптиміст)

є (сміт, що спить)

Ці факти виражають одиничні відносини, зазначені ліворуч від дужок, і перераховані в дужках деякі об'єкти, що зв'язуються даним співвідношенням.

У вирахуванні предикатів іменами відносин відповідає термін предикати, а об'єктам – аргументи. Порядок аргументів повинний завжди задаватися відповідно до інтерпретації предиката, прийнятого в рамках предметної області. Предикат може мати довільне число аргументів. Окремі висловлення можуть об'єднуватися в складні висловлення за допомогою логічних відносин И (and, &), АБО (or, V), НЕ(not, ~) і імплікація (>). Використаний вище метод висновку носить латинську назву modus ponens (модус поненс або скорочення твердження). Його можна записати в такий спосіб:

Якщо істинно А і істинно А > В, то можна вивести В.

Знак > імплікація (вимовляється влече) застосовується для формування правил і читається «ЯКЩО ...., ТО .....». Таким чином, із двох даних пропозицій можна вивести третю.

Деякий об'єкт може бути представлений як константа, тобто як конкретний індивідуум, або як перемінна, наприклад:

є (Х, фахівець з ЕОМ).

Коли перемінній ставиться у відповідність визначене ім'я деякого об'єкта, тобто деякої константи, відбувається породження екземпляра цієї перемінної. Тут

усі константи починаються з малої літери. Для того щоб у вирахуванні предикатів можна було маніпулювати перемінними, треба було ввести додаткову структуру-квантор.

Квантори служать для вказівки міри, у яку екземпляри перемінних повинні бути щирими, для того щоб у цілому висловлення стало щирим. Розрізняють квантор спільності, що позначається символом ∀ - квантор спільності, що означає "для кожного" або "для всіх", і квантор існування, якому відповідає символ ∃.

Квантор ∃ застосовується, коли потрібно сказати, що існує хоча б одне деяке значення перемінної, для якої істинно дане твердження.

Якщо застосовується квантор спільності, то говорять, що твердження щире для всіх X у деякій безлічі. Іншими словами, якщо підставити замість X будь-яке значення, то твердження буде істинно.

Якщо у твердження входять кілька кванторів, то приходиться враховувати їхнє взаємне розташування.

Користуючись кванторами, можна представити пропозицію:

Усі фахівці з ЕОМ є програмістами.

у такий спосіб:

∀ *(X) фахівець з ЕОМ (X) →програміст(X).*

Пропозиція

Деякі фахівці з ЕОМ є оптимістами.

може бути представлене так:

∃ *(X) фахівець з ЕОМ (X) →оптиміст(X).*

Порядок, відповідно до якого вводяться квантифіцируючі перемінні, може впливати на зміст твердження: Наприклад, вираження

∀ *(X)* ∃ *(Y) службовець (X) →керівник(Y,X).*

Може бути інтерпретоване так:

У кожного службовця є керівник.

Якщо ж змінити порядок проходження кванторів, наприклад

∃ ∀ *(X)* ∀ *(Y) що служать (X) →керівник(Y,X).*

то зміниться і твердження

Є така особа, що керує усіма.

У вирахуванні предикатів існує багато різних правил висновку. Вони можуть застосовуватися або для встановлення істинності твердження в цілому, або для породження висновку. От деякі правила висновку:

1. Modus ponendo ponens (MPP): A→ B, A ├ B (├ означає "вірно, що "або "має місце").
2. Modus tollendo tollens (MTT): A→ B, ~A ├ ~B (наприклад ЯКЩО моя програма правильна, ТО вона буде працювати, моя програма НЕ буде працювати, ОТЖЕ, вона не правильна).
3. Подвійне заперечення (ДЗ): ~A ├ ~(~ B) (наприклад, моя програма працювала ОТЖЕ, моя програма НЕ НЕ працювала).
4. Уведення кон'юнкції (УК): A, B ├ {A & B} (наприклад, моя програма працювала, вона правильна, ОТЖЕ, моя програма працювала И вона правильна).
5. Reductio ad absurdum (RAA): A→ B, A→ ~ B ├ ~A (наприклад, ЯКЩО моя програма правильна, ТО вона буде працювати, ЯКЩО моя програма правильна ТО вона не буде працювати, ОТЖЕ, моя програма не правильна).
6. Спеціалізації (С): ∀ (X) W(X), A ├ W(X) (наприклад, усі предмети, що є комп'ютерами, не надійні, «TIPTOP»- комп'ютер, ОТЖЕ, «TIPTOP» ненадійний).

Розкриваючи дужки і приводячи подібні члени, завжди можна привести формулу вирахування висловлень до однієї з двох форм:

1. Кон'юнктивна нормальна форма. Формула записана у виді кон'юнкції диз'юнктів (С1 і С2 і... CN). Кожен диз'юнкт є диз'юнкцією атомарних пропозицій або заперечень таких пропозицій (Р1 АБО НЕ Р2 АБО ... РМ). Це допомагає в міркуваннях, тому що істинність усієї формули означає істинність кожного диз'юнкта окремо, що полегшує розгляд формули.

2. Диз'юнктивна нормальна форма. Формула записана у виді диз'юнкції виражень, кожне з яких є кон'юнкцією атомарних пропозицій або їхніх заперечень. У такій формі часто буває зручно записувати булеві вираження, що

описують перемикальні схеми або умови настання яких-небудь подій. Нехай, наприклад, ми хочемо написати, що два члени комітету, що складаються з трьох чоловік (Андерсон, Бейнс і Кларк), голосували за деяку пропозицію, а один голосував проти. Ми можемо символічно записати "Андерсон голосував "за" у виді А, "Бейнс голосував "за"- у виді В, і "Кларк голосував проти" у виді "НЕ С". Виражаючи цю можливість і дві інші у виді складеної пропозиції в диз'юнктивній нормальній формі, можна написати

(А І В І НЕ С) АБО (НЕ А І В І С) АБО (А І С І НЕ В).

Кожне з виражень у дужках істинно лише при одній конкретній комбінації значень А, В и С. Для будь-яких значень А, В и С, відмінних від цих комбінацій, усі вираження в дужках будуть помилковими, так що і формула в цілому буде помилковою. Таким чином, якщо ця формула вірна для голосування в комітеті, то твердження А, В, С не можуть бути одночасно щирі, не можуть бути одночасно помилкові і більш ніж одне з них не може бути помилковим.

2.2. Доказ із уведенням допущення

Для доказу імплікації виду $A \rightarrow B$ допускається, що ліва частина А щира, тобто А приймається як додаткова посилка, і робляться спроби довести праву частину, В.

Сам метод спирається на дві важливі теореми про докази [50].

Теорема 1. $A \vdash B$ тоді і тільки тоді, коли $\vdash A \rightarrow B$.

Ця теорема затверджує, що довідність висновку В з допущення А еквівалентна довідності імплікації $A \rightarrow B$ без яких-небудь додаткових допущень.

Теорема 2. $A1,A2,...,An \vdash B$ тоді і тільки тоді, коли

$$\vdash (A1 \wedge A2 \wedge ... \wedge An) \rightarrow B.$$

Ця теорема виходить з попереднього і того факту, що всі посилки A1,...,An щирі тоді і тільки тоді, коли щира їх кон’юнкція (основна властивість зв'язування І).

Нарешті, дуже корисна еквівалентність $\models (X \rightarrow (Y \rightarrow Z)) \leftrightarrow (X \wedge Y \rightarrow Z)$.

Вона легко доводиться за допомогою співвідношень булевої алгебри тому що ліва і права частини зводяться до ~X∨ ~Y∨ Z.

Розглянемо приклад доказ з уведенням допущення.

Якщо A1. A2,..., An, P ├ Q,

то ├A1∧ A2∧ …∧AnP∧ →Q) у силу теореми 2,

відкіля ├A1∧ A2∧ …∧An)→(P→Q) у силу еквівалентності,

відкіля ├A1,A2,...,An ├P→Q) у силу теореми 2.

2.3. Доказ приведенням до протиріччя

При побудові висновків не завжди доцільно чекати появи шуканого висновку, просто застосовуючи правила висновку. Саме таке часто трапляється, коли робиться допущення B для доказу імплікації B>C. Застосовуємо ланцюгове правило і модус поненс до B и інших посилок, щоб наприкінці одержати C. Однак можна піти по неправильному шляху, і тоді буде доведено багато пропозицій, більшість з яких не мають відносини до нашої мети. Цей метод зветься прямої хвилі і має тенденцію породжувати лавину проміжних результатів, якщо його запрограмувати для комп'ютера і не обмежити глибину.

Інша можливість - використовувати одну з приведених вище еквівалентностей і спробувати, наприклад, довести ~C>~B замість B>C. Тоді допустимо ~C і спробуємо довести ~B. Т.е. допускається, що висновок C (права частина вихідної імплікації) невірно, і робиться спроба спростувати посилку B. Це дозволяє рухатися як би назад від кінця до початку, застосовуючи правила так що старий висновок відіграє роль посилки. Така організація пошуку називається *пошуком від мети.*

Можна використовувати також комбінацію цих методів, називану приведенням до протиріччя. У цьому випадку для доказу B→C допускаємо одночасно B і ~C, тобто припускаємо, що висновок помилковий:

$$\sim(B \rightarrow C)=\sim(\sim B \vee C)=B \wedge \sim C.$$

Тепер можна рухатися і уперед від В, і назад від ~С Якщо С виведене з В. те, допустивши В, довели б С. Тому допустивши С. одержимо протиріччя. Якщо ж ми виведемо ~В з ~С, то тим самим одержимо протиріччя з В. У загальному випадку ми можемо діяти з обох кінців, виводячи деяку пропозицію Р, рухаючи вперед, і його заперечення ~Р, рухаючи назад. У випадку удачі це доводить, що посилки несумісні, або суперечливі. Звідси виводимо, що додаткова посилка ($B \wedge \sim C$) повинна бути помилкова, а значить протилежне їй твердження ($B \wedge C$) істинно.

Слід зазначити, що якщо пропозиція B>C ложно, то ніколи не одержимо протиріччя, скільки б ні міркували. При автоматизації пошуку доказу тут знову виникає проблема, тому що ЕОМ не може сказати, коли потрібно зупинитися.

2.4. Доказ методом резолюції

Застосовується всього одне правило висновку, що дозволяє не запам'ятовувати численних правил висновку і тавтології. Це - правило резолюції, що приведено в таблиці 2.1 разом із уже відомими правилами.

Таблиця 2.1

Резолюція	Ланцюгове правило	Модус поненс
З $X \vee A$	З $\sim X \rightarrow A$	А

I Y∨ ~A	I A→Y	A→Y
Одержуємо X∨ Y	Одержуємо ~X→Y	Y

Воно дозволяє з'єднати дві формули, видаливши з однієї атом A а з іншої ~A. З приведеної вище таблиці видно, що правило резолюції можна розглядати як аналог ланцюгового правила в застосуванні до формул, що знаходяться в кон'юнктивній нормальній формі В даному випадку ланцюгове правило записане у формі, еквівалентній приведеній раніше, але більш зручної для зіставлення з правилом резолюції. Правило модус поненс можна вважати частковим случаємо правила резолюції для випадку помилкового X.

Правило резолюції застосовуємо в такий спосіб.

Використовуємо доказ від противного і допускаємо заперечення висновку.

Приводимо всі посилки і заперечення висновку, прийнятого як додаткову посилку, до кон'юнктивної нормальної форми.

а) Усуваємо символи ↔і→ за допомогою еквівалентностей

(B) = (A→B) ∨ C)∧ (B→A);

A→B = ~A∨ B

б) Просуваємо заперечення усередину за допомогою закону де Моргана.

в) Застосовуємо дистрибутивність

A∨ (B∨ C)=(A∨ B)∧ (A∨ C).

Цей метод приведення формул до виду, зручний для застосування методу резолюції, володіє поруч серйозними недоліками, тому що застосування дистрибутивності, часто приводить до експонентного розбухання- випробуваної формули і руйнує її структуру. У пропозиціональному випадку краще привести до

кон’юнктивної нормальної форми саму вихідну формулу, що відразу вирішує питання про її тавтологічність.

Найчастіше набагато доцільніше спрощувати формулу до виду, поступово зменшуючи її глибину шляхом уведення нових атомів, що позначають її частини.

Тепер кожна посилка перетворилася в кон’юнкцію диз’юнктів, бути може, одночленну. Виписуємо кожен диз’юнкт із нового рядка; усі диз’юнкти щирі, тому що кон’юнкція щира по припущенню.

Кожен диз’юнкт - це диз'юнкція (можливо, одночленна), що складається з пропозицій і заперечень пропозицій. Саме до них застосуємо метод резолюцій. Беремо будь-які два диз’юнкта, що містять той самий атом, але з протилежними знаками, наприклад

$$X \vee Y \vee Z \vee \sim P, \quad X \vee P \vee W.$$

Тут P - саме той атом, про який йшла мова. Застосовуємо правило резолюції з атомом P в ролі A з цього правила, тобто "відрізаємо" P від двох даних диз’юнктів:

$$(X \vee Y \vee Z) \vee (Y \vee W) = X \vee Y \vee Z \vee W.$$

Це правило легко застосовувати, тому що воно зводиться до простого видалення членів диз’юнктів.

З $X \vee A, \sim A \vee Y$ виводимо $X \vee Y$.

З $A, \sim A \vee Y$ виводимо Y.

Продовжуємо цей процес, поки не вийде P и ~P для деякого атома P. Застосовуючи резолюцію і до них, одержимо порожній диз’юнкт, що виражає протиріччя, що завершує доказ від противного.

З P, ~P виводимо неправду.

Порожній диз’юнкт звичайно записується у виді квадрата ?.

Як приклад розглянемо доказ співвідношення:

$P \vee Q,\ P \rightarrow R,\ Q \rightarrow S \vdash R \vee S$

1. Приводимо посилки до нормальної форми і виписуємо їх на окремих рядках.

$P \vee Q,$ (2.12)

$\sim P \vee R,$ (2.13)

$\sim Q \vee S$ (2.14)

2. Записуємо заперечення висновку і приводимо його до нормальної форми.

$\sim(R \vee S) = \sim R \wedge \sim S$

$\sim R$ (2.15)

$\sim S$ (2.16)

3. Виводимо порожній диз’юнкт за допомогою резолюції.

~P з (2.15) і (2.13) (2.17)

Q з (2.17) і (2.12) (2.18)

~Q з (2.16) і (2.14) (2.19)

□ з (2.18) і (2.19) (2.20)

У порівнянні з класичною логікою метод резолюції має ряд переваг.

- Не приходиться застосовувати еквівалентності для того, щоб переставляти члени диз'юнкції $P \vee Q$ для одержання $Q \vee P$ і т.д. Це відбувається почасти тому, що усе приводиться до нормальної форми із самого початку, а почасти тому, що для правила резолюції неважливе положення атома, що відрізається, у диз’юнкті.

- Потрібно пам'ятати усього одне правило.

Застосування цього правила легко автоматизувати, тобто запрограмувати для комп'ютера. Однак у випадку довгого доказу легко зациклитися, а однорідні позначення роблять усі диз'юнкти схожими один на одного, так що важко утримувати в пам'яті їхній зміст і вибирати потрібний диз'юнкт.

2.5. Застосування методу резолюцій для відповідей на питання

Припустимо, що предикат F(x,y), означає x - батько y, і дані наступні факти про батьківство:

$F(john,. harry) \wedge F(john, sid) \vee F(sid, liz)$

Таким чином, є три одиничних (тобто складаються з однієї літери) диз'юнкта. Вони не містять перемінних або імплікацій, а просто представляють базисні факти того самого типу, що підходить для збереження в базі даних.

Уведемо ще три предикати M(x), S(x,y) і B(x, y), що означають відповідно, що x - чоловік, що він єдинокровен з y, що він брат y.

$(\forall\ x, y)\ F(x, y) \rightarrow M(x)$

$(\forall\ x, y, w)\ F(x,y) \wedge F(x, w) \rightarrow S(y, w)$

$(\forall\ x, y) S(x, y) \wedge M(x) \rightarrow B(x, y)$

Вони затверджують наступне: 1) усі батьки - чоловіки; 2)якщо в дітей один батько, то вони єдинокровні; 3) брат - це єдинокровний чоловік.

Нехай поставили запитання $(\exists\ z)\ B(z, harry)$? Щоб знайти відповідь за допомогою методу резолюції, записуємо заперечення питання $(\forall\ z) \sim B(z, harry)$. Потім приводимо аксіоми до нормальної форми і записуємо кожен диз'юнкт в окремому рядку (тому що кожен диз'юнкт щирий сам по собі):

$$\sim P(x, y) \vee M(x) \qquad (2\ 21)$$

~P(x, y)∨ ~F(x, w)∨ S(y, w) (2 22)

~S(x, y)∨ ~M(x)∨ B(x,y) (2.23)

F(john, harri) (2.24)

F(john, sid) (2 25)

F(john, liz) (2.26)

~B(z, harri) (2.27)

Не пишемо зовнішніх кванторів загальності, тому що мається на увазі, що кожна перемінна зв'язана таким квантором.

Константи відрізняються від перемінних тим, що вони або починаються буквами з початку латинського алфавіту, або є іменами власними, такими, як john, harri, sid і liz.

Для застосування резолюції необхідно знайти для даної пари диз'юнктів таку підстановку термів замість перемінних, щоб після неї деяка літера одного з диз'юнктів стала контрарна деякій літері з іншого диз'юнкта (тобто відрізнялася від неї лише запереченням). Можна робити підстановки, тому що всі перемінні зв'язані кванторами загальності. Якщо, наприклад, підставимо john замість x і sid замість y, то одержимо наступне:

~ F(john, sid)∧~ F(john, w)∧S(sid, w).

Можна застосувати правило резолюції до цьому диз'юнкту і (2.25), що дає новий диз'юнкт:

F(john, w)∨ S(sid,w) (2.28)

з (2.5) і (2.2) {john/x, sid/y}

У коментарі праворуч у фігурних дужках зазначені два дизъюнкта використані в резолюції, і застосована підстановка.

Продовжуючи, одержимо

S(sid, harri) (2.29)

з (2.24) і (2.27) {harri/y}

M(sid) (2.30)

з (2.26) і (2.21) {sid/x, liz/y}

~S(sid, y)∨ B(sid, y) (2.31)

з (2.30) і (2.23) {sid/x}

B(sid, harry) (2.32)

з (2.29) і (2.31) {harry/y}

?

з (2.22) і (2.27) {sid/z} (2.33)

Таким чином, вивели одиничний диз'юнкт (2.22), який виражає що sid - брат harry, використовувавши аксіоми і факти (2.24), (2.25), (2.26), що маються в базі даних. Це суперечить запереченню питання, яке затверджує, що harry не має братів.

Якщо хочемо знати, хто ж брат haггy, то повинні стежити за зробленими підстановками. Це можна зробити, увівши нову літеру – так називану літеру відповідь із предикатом Answer(відповідь).

Тоді замість ~B(z, harri) напишемо

~B(z, harri) ∨ Answer(z). (2.27a)

Тепер продовжимо процес резолюції не до одержання порожнього диз'юнкта, а до одержання одиничного диз'юнкта з предикатом Answer, що запам'ятовує підстановку для z, потрібну для резолюції з диз'юнктом B(z, harri). Тоді (2.33) прийме вид Answer(sid) з (2.22) і (2.27a) {sid/z}.

Отримана відповідь означає, що sid – брат harry. При іншій вихідній інформації одержали б замість одиничного диз'юнкта, скажемо Answer(sid)∨ Answer(fred).

Навіть якщо провести резолюцію в трохи іншому порядку, то одержимо ту ж саму відповідь. Допустимо, що дійшли до рядка (2.21) так само, як і раніш, але потім застосували (2.27a) замість (2.27).

~S(x, harry)∨ ~M(x)∨ Ansver(x) (2.21)

з (2.27a) {x/z} і (2.23)

{harry/y} (2.21)

S(sid, harri)∨ Answer(sid) (2.22)

з (2.30) і (2.31) {sid/x}

Answer(sid) (2.33)

з (2.32) і (2.29)

Однак якщо робити підстановки в іншій послідовності, то попадаємо в скрутне положення:

~ F(john, w)∨ S(sid,w) (2.28)

з (2.26) і (2.22) {john/x, harry/x)

S(harry, sid) (2.29)

з (2.25) і (2.28) {sid/w}

Проблема в тім, що довели S(haггy, sid) замість диз'юнкта S(sid, haггy), що потрібний нам для резолюції з (2.32). Ці труднощі можна обійти, увівши додаткову аксіому:

(∀ x, y) S(x, y) →S(y, x),

яка в нормальній формі має вигляд

$\sim S(x, y) \vee S((y, x)$ (2.34)

На жаль, далі потрібно виявляти обережність, щоб не потрапити в наступний нескінченний цикл:

S(sid, harry) (2.35)
з (2.29) і (2.34) {harry/x, sid/y}
S(harry, sid) (2.36)
з (2.35) і (2.34) {sid/x, harry/y}
S(sid, harry) (2.37)
з (2.36) і т.д.

Це показує, що необхідно ретельно планувати пошук доказу, якщо хочемо зберегти надію шукати його автоматично.

2.6. Евристики для пошуку доказу

При написанні програми, що буде робити за нас резолюцію і шукати відповіді на питання, корисно використовувати наступні методи.

Перевага одиничних. Найкраще, коли хоча б один з приймаючих участь у резолюції диз'юнктів - одиничний, тому що при цьому уміщається довжина результуючого диз'юнкта. Якщо замість цього застосуємо резолюцію, наприклад, до двох диз'юнктів, кожний з яких містить по три літери, то одержимо результат, що містить чотири літери, і так далі, що веде нас ще далі від простого диз'юнкта.

Стратегія підтримки. Для одержання відповіді на конкретне питання варто використовувати конкретні факту з постановки задачі, а не тільки загальні аксіоми. Можна прийняти, наприклад, резолюцію по диз'юнктам (2.21) - (2.23) і одержувати усе більш загальні теореми про сімейні відносини, але це ніколи не дало б конкретної відповіді. Для її одержання прийдеться використовувати

диз’юнкти (2.24) - (2.26) і їх наслідок. Простежуючи попередників даного диз’юнкта у висновку, можна це забезпечити.

Вибір унікальних літер. У приведеному вище прикладі єдине входження предиката B без заперечення знаходиться в диз’юнкті (2.23), тому ясно, що саме до нього повинна застосовуватися резолюція, у якій бере участь ~B з (2.27). Навпроти, ~F має багато входжень, тому краще почати з резолюції по літерах начебто B. щоб одержати більш короткі диз’юнкти, де замість частини перемінних зроблені підстановки, що дають навідні розуміння для подальших підстановок.

Планування. Людина звичайно намагається спланувати доказ, малюючи діаграми і роблячи начерки доказу перш ніж виписати все докладно. Може виявитися, що необхідно аксимотизувати інформацію про те, які аксіоми найбільш корисні. Це - метаінформація. Такий прийом був успішно застосований до Прологу для того, щоб проводити доказ в “просторі планування”, а потім використовувати результати для побудови кістяка доказу і вибору використаних аксіом і підстановок, а також їхнього порядку.

2.7. Підстановка й уніфікація

Метод резолюції вимагає, щоб робилися підстановки в диз’юнкти так, щоб дві літери з протилежними знаками містили співпадаючі атоми. Тому що ці атоми можуть містити перемінні, можна робити і подальші підстановки замість них, а літери як і раніше будуть відрізнятися тільки знаками. Процес перебування найбільш загальної підстановки, що робить дві літери контрарними, називається уніфікацією. Розглянемо уніфікацію двох літер.

P(a, x, f(g(y)) і ~P(z, f(z),f(u)).

Поступово будуємо підстановку S. яка являє собою набір пар виду (t/x), де t - терм, x - перемінна, замість якої він буде підставлений. От ще приклад: S={harry/x, z/w}. У загальному випадку пишемо S={t1/v1, t2/v2,...}, де vi - попарно різні перемінні і кожен терм ti відмінний від своєї перемінної vi. Йдемо ліворуч

праворуч, відшукуючи пари термів, що не збігаються один з одним. Так послідовно одержуємо:

Сполучити P(a,x,f(g(y)) з ~P(z,f(z),f(u))

S={a/z} Сполучити P(a,x,f(g(y)) з ~P(a,f(a),f(u))

S={a/z,f(a)/x} Сполучити P(a,f(a),f(g(y))) з ~P(a,f(a),f(u))

S={a/z,f(a)/x,g(y)/u} Сполучити P(a,f(a),f(g(y))) з ~P(a,f(a),f(g(y))).

Цей метод дає найбільш загальний уніфікатор. Були зроблені підстановки замість z,x, і u але не змінювали в. Більш обмежувальна підстановка може мати вигляд

S={a/z, f(a)/x, g(y)/u, a/y}, що дає P(a, f(a),f(g(a))).

Цей процес відіграє важливу роль у мові Пролог, тому що він узагальнює механізм виклику процедур, використовуваний у звичайних мовах програмування. Звичайно аргументи виклику процедури - це вираження, що підставляються замість формальних параметрів і можуть бути тільки іменами перемінних. Однак у Пролозі дозволяється, щоб самі формальні параметри були термами, і тому процес виклику "логічної процедури" включає сполучення термів, що є аргументами виклику, з термами з заголовка процедури за допомогою методу уніфікації. Якщо уніфікація закінчується невдачею через те, що ніяка підстановка не зможе сполучити потрібні літери, то виклику не відбудеться, але буде зроблена спроба сполучення з іншим визначенням процедури, якщо такі маються.

Вирахування предикатів дає можливість виразити багато чого з того, про що хотілося б говорити або міркувати. Використовуючи предикати, квантори, перемінні і функціональні символи, можна виразити досить складні твердження; прості факти записати у виді атомів, що складаються з предикатів з константами як аргументи. Таким чином, можна виражати різні знання, як загальні аксіоми, так і факти, у рамках єдиного формалізму, що важливо для експертних систем.

При автоматизації висновку доказів методами вирахування предикатів потрібно визначити ряд процедур для вибору правил, що дозволяють запобігти комбінаторний вибух і забезпечити проведення немонотонних міркувань. При цьому намітилося два підходи.

При першому підході до рішення задачі керування, що нашли широке поширення в додатках концепції штучного інтелекту, відкидався принцип універсальності і вироблявся пошук методів обробки даних, ефективних для рішення конкретних задач у межах визначеної предметної області. Джерелом високої ефективності цих методів служать введені в систему великі знання в конкретній предметній області і проходження фактів, які спостерігаються, що полягає в тім, що людина, що виступає в ролі експерта - вирішувателя задач, дуже рідко користується процедурами загального характеру, досягаючи успіху за рахунок гарного “знання” змісту задачі. З метою спростити процес програмування для таких “заснованих на знаннях” систем були розроблені відповідні формалізми, що володіють меншими виразними можливостями, чим вирахування предикатів, але більш зручні для реалізації на ЕОМ. Багато подібних прийомів були розроблені з використанням засобів мови обробки списків - Лісп.

Другий підхід, що розвивається в рамках традиційної логіки, був спрямований на збереження універсальності, властивої вирахуванню предикатів, шляхом виявлення процедур висновку доказів, універсальних за своїм характером і в той же час що дозволяють обійти проблему вибору правил і комбінаторного вибуху. Цей підхід в основному становить інтерес для самої логіки, зокрема, при розробці методів автоматизації висновку доказів (названого “доказом теорем”). Для цього і була розроблена мова програмування Пролог, що представляє собою комбінацію одного з методів доказу теорем - “методу резолюції” і винятково ефективної стратегії керування.

Побачили також, яким образом твердження вирахування предикатів приводяться до нормальної форми і як використовувати метод резолюції для доказу їхньої істинності і перебування значень перемінних (зв'язаними

кванторами існування), що задовольняють цим твердженням. Ці методи складають також основу мови Пролог.

Контрольні питання

1). Вивчіть наступні правила висновку:

A →m	b → p	d i a → n
F → y	b i c → l	x i a → z
E → p	c i m → x	p i y → w

2). Відомо, що a, b i c вірно. Доведіть істинність z і w.

3). Напишіть факти правила за допомогою яких можна установити ЯКЩО A і B учаться в одній групі ТО знають один одного.

4). Напишіть факти правила за допомогою яких можна установити ЯКЩО A і B чоловік і жінка ТО C i D брат і сестра.

5). Доведіть за допомогою методу резолюцій істинність тверджень:

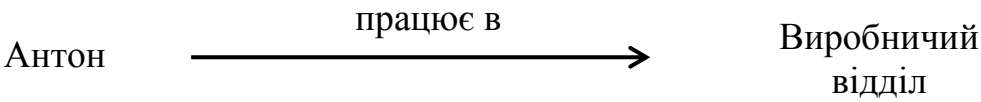
a)	b)
P ∨ Q	~B ∨ A
A ∨ B	C ∨ B
(A ∨ ~P) v(B)	~C
B істинно?	A істинно?

3. Представлення знань предметної області

3.1.Семантичні мережі

Поняття семантичної мережі засновано на древній і дуже простій ідеї про те, що <пам'ять> формується через асоціації між поняттями. Поняття <асоціативна пам'ять> з'явилося ще в часи Аристотеля і ввійшло в інформатику в зв'язку з роботами по використанню простих асоціацій для представлення значення слів у базі даних. З тих пір цей формалізм був усебічно розвитий для представлення багатьох класів даних, використовуваних у різних предметних областях. До таких областей відносяться просторові зв'язки в простих фізичних системах, операції по керуванню механізмами, причинні і функціональні зв'язки в приладах і взаємозв'язку між симптомами в медицині [33].

Базовим функціональним елементом семантичної мережі служить структура з двох компонентів - <вузлів> і єднальних ліній <дуг>. Кожен вузол представляє деяке поняття, а дуга - відношення між парами понять.



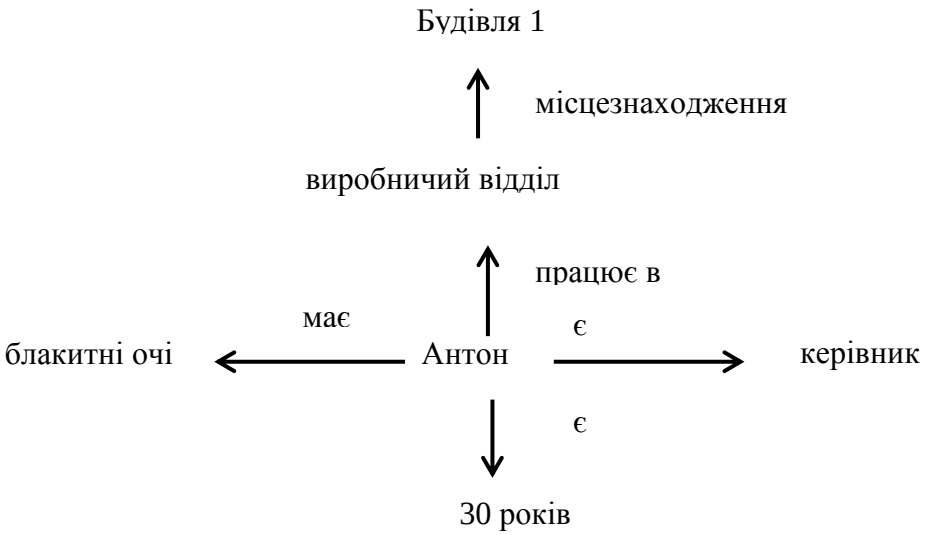
Малюнок 3.1Функціональний елемент семантичної мережі

Можна вважати, що кожна з таких пар відносин представляє простий факт. Вузли позначаються ім'ям відповідного відношення. Рис. 3.1, наприклад, представляє факт <Антон працює у виробничому відділі>.

Відзначимо, що дуга має спрямованість, завдяки чому між поняттями, у рамках визначеного факту, виражається відношення <суб'єкт/об'єкт>. Більш того, будь-який з вузлів може бути з'єднаний з будь-яким числом інших вузлів; у результаті цього забезпечується формування мережі фактів.

З позицій логіки базову структуру семантичної мережі можна розглядати як еквівалент предиката з двома аргументами (бінарний предикат); ці два аргументи

представляються двома вузлами, а власне предикат - спрямованою дугою, що зв'язує ці вузли. Користуючись подібними відносинами, можна представляти складні сукупності фактів. Рис. 3.2 ілюструє одне з можливих представлень



Малюнок 3.2 Факти про людину Антон

фактів, що відносяться до службовця <Антон>. У число таких фактів увійшли:

< Антон є керівником >

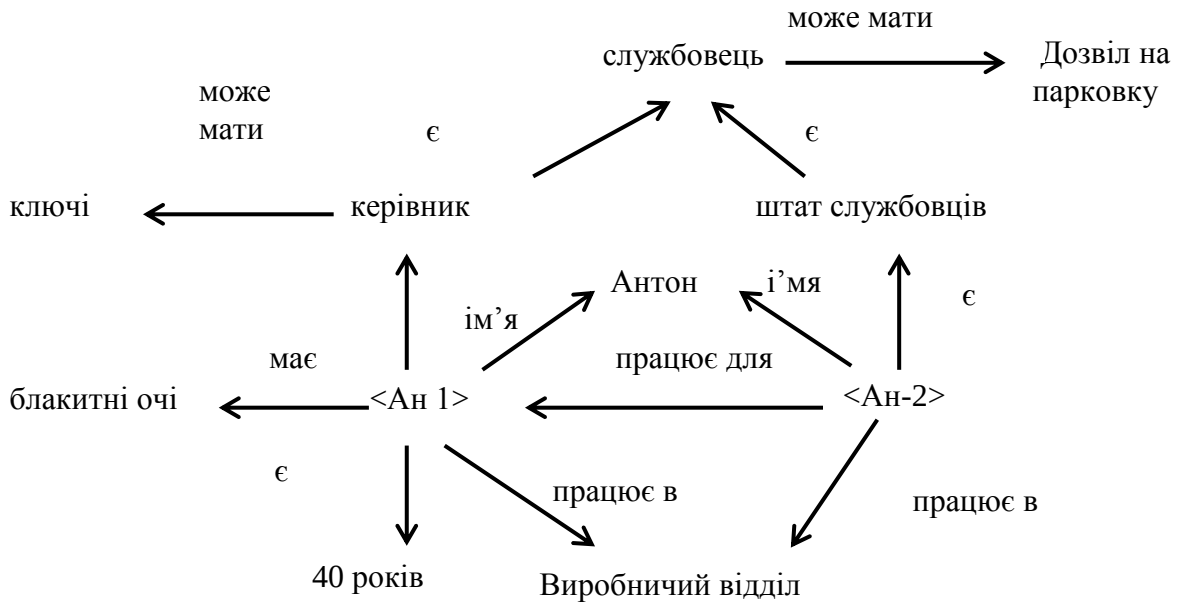
< Антон працює у виробничому відділі, розташованому в будинку 1>

< Антону 40 років >

<В Антона блакитні очі >

Давня популярність семантичних мереж зобов'язана зв'язку <є>, у якому закладені великі можливості для побудови ієрархій понять. Приклад такої ієрархії вузол <службовець> на Малюнок 3.3. Вузли <Ан-1> і <Ан-2> дозволяють описати двох різних людей з однаковим іменами. Ієрархія, побудована на основі спадкування, забезпечує ефективний спосіб спрощення представлення знань і скорочення обсягу інформації котру потрібно запам'ятовувати для кожного конкретного вузла. Це дає можливість значною мірою прискорити процес обробки знань (що відноситься до вузла інформація, що запам'ятовується, може бути обмежена

тільки часто використовуваної; при звертанні до іншої інформації застосовується принцип спадкування), а також витягати інформацію за допомогою запитів загального характеру (деяка інформація про індивідуума <Антон> як керівнику може



Малюнок 3.3 Факти, що відносяться до службовця <Антон>

бути витягнута просто зі знання його службового становища в компанії; при цьому немає необхідності знати його ім'я).

Приведені приклади представлення знань у виді семантичних мереж були обмежені відносинами між іменниками або фразами, складеними з іменників. Потрібно тільки розробити представлення щодо дієслів на додаток до іменників.

Були розроблені несуперечливі повні набори вербальних відношень. Вони одержали назву <відмінкових відносин > відповідно до теорії <відмінкової граматики >, розробленої Ч. Філмором. У цій теорії починається спроба представити поверхневу структуру пропозиції у виді невеликих замкнутих наборів <відмінкових> відношень між іменниками (або фразами з іменників) і дієсловами в рамках глибинних структур пропозицій. Зразковий набір подібних відношень може в типовому випадку включати:

а) Агент - виконавець (ініціатор) дії, що виражається дієсловом;

б) Об'єкт - ім'я іменник, на яке поширюється дія або стан, що виражається дієсловом;

в) Місцеположення-місце дії або стану виражається дієсловом;

г) Датив (dative) - особа, до якої має відношення дія або стан, що виражається дієсловом.

3.2. Фрейми

Принцип організації властивостей деякого об'єкта або події для формування прообразу реалізується за допомогою нотації виду <фрейм>. Для ілюстрації цього опишемо зведення про службовця компанії, як це зроблено в попередньому розділі. Достоїнство системи, що використовує фрейми, полягає в тім, що ті елементи, що традиційно присутні в описі об'єкта або події, групуються і завдяки цьому можуть витягатися й оброблятися як єдине ціле. Перший приклад стосується поняття <керівник> (див. мал. 3.4) і ілюструє деякі особливості фреймів.

ім'я: КЕРІВНИК
спеціальність: СЛУЖБОВЕЦЬ
ім'я: _____
вік:_____
адреса: _____
відділ:_____
заробітна плата:_____
дата початку: _____
до: _____

Малюнок 3.4 Кістяковий фрейм для поняття <КЕРІВНИК>

По-перше, фрейм має ім'я для ідентифікації описуваного їм поняття. По-друге, його опис складається з ряду описів, приведених на малюнку ліворуч, що

одержали назву <слоти>. За допомогою слотів ідентифікуються основні структурні елементи понять. За слотами випливають шпації (проміжки), у які поміщають деякі об'єкти, що представляють поточні значення слотів.

ім'я: КЕРІВНИК
спеціальність: СЛУЖБОВЕЦЬ
ім'я: агрегат (прізвище, ім'я, по батькові)
вік; агрегат (роки)
адреса: АДРЕСА
відділ: діапазон (виробництво, адміністрація)
заробітна плата: ЗАРПЛАТА
дата початку: агрегат (місяць, рік)
до: агрегат (місяць, рік) (за замовчуванням: тепер)

Малюнок 3.5 Фрейм для загального поняття <КЕРІВНИК>

На мал. 3.5 даний той же фрейм, що і на мал. 3.4, але тільки з заповненими слотами. При цьому частина з них заповнена якимись об'єктами, а не простими іменами. У даному прикладі фігурують три різних типи таких заповнювачів слотів. Заповнювач слоту може бути або константою, або ім'ям іншого фрейму. Найпростішими з них є ті, що представлено прописними буквами (наприклад, АДРЕСА, ЗАРПЛАТА). Це імена інших фреймів даної системи, на які робиться посилання. Крім того, існують позначення <агрегат> і <інтервал>. У процесі обробки систем фреймів іноді необхідно накласти обмеження на тип об'єкта, що може бути використаний для заповнення деякого слоту. Позначення <агрегат> указує на те, що повинні бути задані визначені об'єкти, а позначення <діапазон> - на те, що повинен бути обраний один з безлічі об'єктів.

Фрейми порівняно легко реалізуються за допомогою структури мови Лісп у виді списку властивостей, розглянутого раніше в зв'язку зі звертанням до семантичних мереж.

Існує ряд мов, спеціально розроблених для полегшення конструювання на основі фреймів різних процедур обробки знань.

3.3. Правила продукцій

Найпоширенішим форматом для представлення знань, найбільш відповідному їхньому процедурному характерові, є правило продукції, що по своїй суті - просто програма з одного оператора виду:

<ЯКЩО умова, ТО дія>

Нотація процедур у виді послідовностей правил була вперше запропонована математиком Постом.

Люди мають запас "знань" про світ, у якому вони живуть. Деякі види знань загальновідомі; до них відносяться знання, зв'язані з прийомом їжі або водінням автомобіля. Інші знання більш спеціальні, наприклад ті, що використовуються експертами. Знання звичайно представляються у виді фактів, характерних для навколишнього світу (тобто класів об'єктів і взаємозв'язків між ними), процедур і правил маніпулювання фактами, а також у виді інформації про те, коли і як варто застосовувати правила і процедури.

Об'єкти групують по класах, Петро, Джон, Фред і Ганна можуть мислитися як об'єкти. Їх можна віднести до класу “особистість”. На додаток до цього Петро, Джон і Фред можуть бути класифіковані як “чоловіки”, а Ганна - як "жінка". Явне достоїнство будь-якої класифікації полягає в тім, що частково вирішується проблема переповнення пам'яті, тому що досить пам'ятати тільки характеристики класу, а не кожного об'єкта. Ми можемо також визначити відносини між класами (або окремими об'єктами). Подібним чином ми можемо визначити відношення “керує (А, В)”, що означає, що В перебуває в підпорядкуванні від А, Як приклади такої залежності можуть служити вираження:

керує (Петро, Джон)
керує (Джон, Ганна)
керує (Ганна, Фред)

які укладають у собі структуру “підзвітність” (інше відношення) між об'єктами Петр-джон-анна-фред, Приведений приклад ілюструє відносини між схожими об'єктами. Між об'єктами, що розрізняються, також можуть бути установлені відносини (наприклад, “володіє (Петро, автомобіль)”). Знання про об'єкти і їхні взаємини дозволяють класифікувати ці об'єкти і співвідносити між собою.

Другий тип знання - правила. Вони дають можливість визначити, як вивести нові відмінні риси класу або відносини для об'єктів, раніше не підрозділених на класи. Наприклад, якщо ми визначимо відношення “звітує (В, А)” для того, щоб відзначити, що В підзвітне А (можливо, через інших керівників), то зможемо установити правила:

“звітує (С, А)” є ІСТИНА
ЯКЩО або “керує (А, С)” є ІСТИНА
АБО “керує (А. В)” І “керує (В, С)" є ІСТИНА

Це досить обмежене правило. Воно застосовано тільки для першого або другого рівня підзвітності, але в межах обмеження воно дає нам можливість породити новий приклад відносин “звітує”, що раніше не був відомий. Наприклад, правило дозволяє зробити висновок про те, що “звітує (Ганна, Петро)” і “звітує (Фред, Джон)” суть ІСТИНА. Унаслідок того, що дане правило визначене як дворівневе, воно не може бути застосоване для одержання висновку “звітує (Фред, Петро)” є ІСТИНА. Для цього нам буде потрібно більш могутнє правило, що включає рекурсію:

“звітує (С, А)” є ІСТИНА

ЯКЩО або “керує (А, С)” є ІСТИНА

АБО “керує (А, В)” є ІСТИНА

І “звітує (С, В)” є ІСТИНА

Перша частина приведеного рекурсивного правила відноситься до прямої підзвітності, а друга - до непрямого.

Якщо поставити запитання “звітує (Джон, Петро)?”, то відповіддю буде ІСТИНА, оскільки щира перша частина правила “ЯКЩО”. Питання “звітує (Фред, Петро)?” зажадає більш складної обробки. Табл. 3.1 показує процес подібної обробки з використанням розглянутого правила.

Таблиця 3.1 Приклад відносини “звітує”

№ Питання	Питання	Перше АБО	Друге АБО	Нове питання
1	Звітує (Фред, Петро)?	керує (Петро, Фред)? Неправда	керує (Петро, В)? істина В= Джон	звітує (Фред, Джон)?
2	Звітує (Фред, Джон)?	керує (Джон, Фред)? Неправда	керує (Джон, В)? істина В= Ганна	звітує (Фред, Ганна)?
3	Звітує (Фред, Ганна)?	керує (Ганна, Фред)? Істина		
Тому звітує (Фред, Ганна) є істина Отже:: звітує (Фред, Джон) є істина Отже: звітує (Фред, Петро) є істина, Що служить відповіддю на питання				

Відзначимо рекурсивність обробки відносини “звітує”. Ця процедура досить складна при всій простоті і зрозумілості самого правила.

Третій необхідний компонент процесу обробки знань - керуюча структура, він визначає спосіб застосування різноманітних правил тим самим визначаючи стратегію рішення.

Контрольні питання

1. Представте географічні дані України (столиця(обласний центр, районний центр), широта, довгота, населення, мова, чоловіки, жінки, національність, кількість національність), використовуючи методи представлення даних у виді:

- семантичної мережі;
- продукціоних правил;
- фреймів.

4. Методи стратегії пошуку рішень

Як було показано в главі 1, експертні системи складаються з трьох компонентів:

- бази знань, що містить правила продукцій;

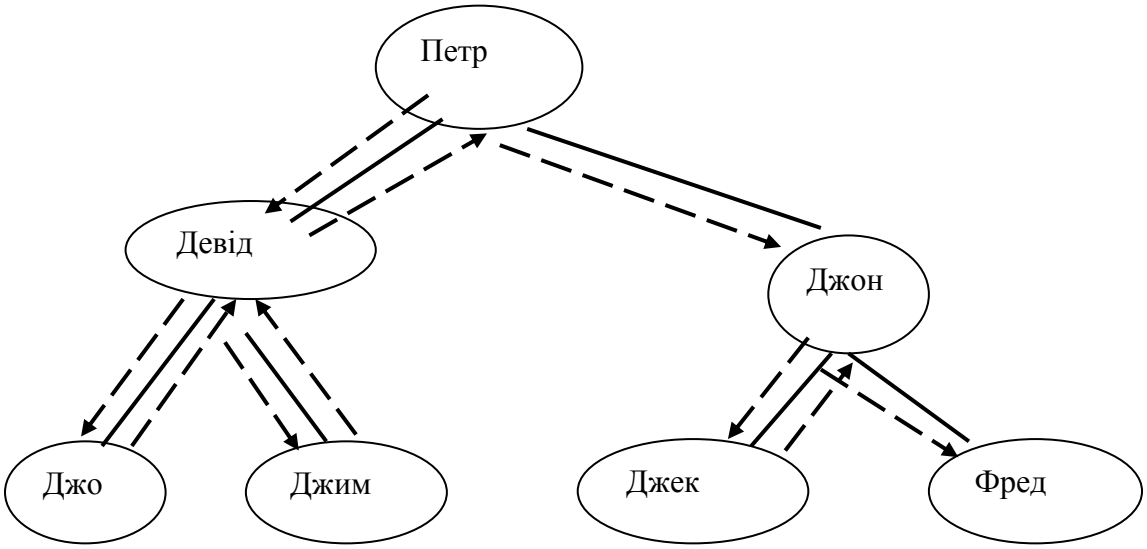
- бази даних, що відображає поточний стан деякої задачі;

- керуючої структури (інтерпретатора), що вирішує, яке з правил продукції слід застосувати наступним.

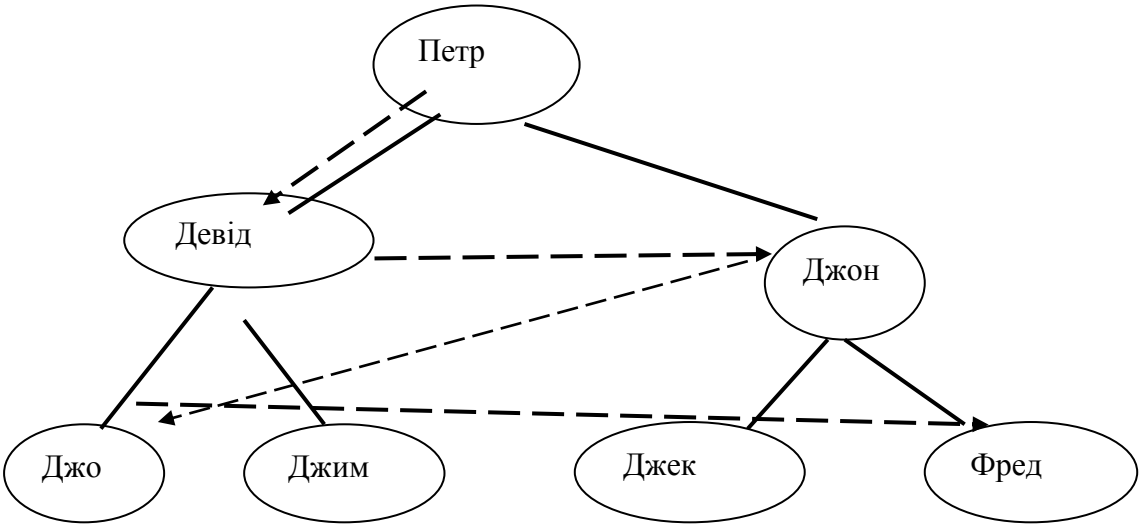
При нормальних умовах в ієрархії керівництва мається безліч подібних шляхів (Малюнок 4.1).

У визначених крапках зіштовхуємося з необхідністю вибору шляхів, для чого потрібен підходящий метод, що задається керуючою структурою. Ясно, що будь-який заснований на логіці підхід забезпечить можливість зробити це, але один з очевидних способів рішення подібної задачі зводиться до того, що спочатку варто рухатися уздовж самої лівої галузі дерева доти, поки не буде, удоволен запит або не буде досягнутий кінець галузі. В останньому випадку потрібно “відступити” у попередню крапку розгалуження і за допомогою того ж методу зробити перевірку інших галузей. У результаті, як це видно (Малюнок 4.1),шлях обробки, що відповідає запитові “звітує (Фред, Петро)?” показаний у виді пунктирної лінії. Це називається пошуком “спочатку всередину”. На цьому ж малюнку зображена і схема пошуку “спочатку вшир”. Обидві стратегії розглянуть усі можливі варіанти, що дуже часто приводить до виникнення комбінаторного вибуху.

На глобальному рівні керування послідовністю застосування правил можна виділити дві стратегії поводження - застосування правил у прямому і зворотному порядку. Прямий порядок означає, що ланцюг міркувань будується, відштовхуючи від відомих фактів до гіпотез. Зворотний ланцюжок означає, що міркування будуються, відштовхуючи від заданої мети до фактів, що підтверджують цю мету.



Пошук "спочатку всередину"



Пошук "спочатку вшир"

Малюнок 4.1 Обробка ієрархії відносин

4.1. Пошук рішення задач у просторі станів

Щоб побудувати опис задачі з використанням простору станів, потрібно мати визначене представлення про те, що собою представляють стани в конкретній задачі. Таким чином, важливим етапом побудови якого - або опису задачі з використанням простору станів є вибір деякої конкретної *форми* опису станів задачі.

Існуючі методи рішення задач, використовувані в експертних системах, можна класифікувати в такий спосіб:

- Методи пошуку в одному просторі – методи, призначені для використання в наступних умовах: області невеликої розмірності, повнота моделі, точні і повні дані;
- Методи пошуку в ієрархічних просторах - методи, призначені для роботи в областях великої розмірності;
- Методи пошуку при неточних і повних даних;
- Методи пошуку, що використовують кілька моделей, призначені для роботи з областями, для адекватного опису яких однієї моделі недостатньо.

4.2. Методи пошуку рішень в одному просторі

Методи пошуку рішень в одному просторі звичайно поділяються на пошук у просторі станів, пошук методом редукції, евристичний пошук і пошук генерація-перевірка [59].

По суті, будь-яка структура величин може бути використана для опису станів. Це можуть бути рядки символів, вектори, двомірні масиви, дерева і списки. Часто обирана форма опису має подібність з деякою фізичною властивістю розв'язуваної задачі. Так, у грі в п'ятнадцять природних форм опису станів може бути масив 4х4. Вибираючи форму опису станів, потрібно подбати і про те, щоб застосування оператора, що перетворить один опис в інший, виявилося б досить легким.

Очевидно, самий прямолінійний підхід при пошуку рішення для гри в п'ятнадцять складається в спробі перепробувати різні ходи, поки не удасться одержати цільову конфігурацію. Такого роду спроба власне кажучи зв'язана з пошуком за допомогою проб і помилок. Відправляючись від початкової конфігурації, можна побудувати всі конфігурації, що виникають у результаті виконання кожного з можливих ходів, потім побудувати наступну безліч конфігурацій після застосування наступного ходу і т.д., поки не буде досягнута цільова конфігурація.

Для обговорення такого сорту методів пошуку рішення введемо поняття *станів* і *операторів* для даної задачі. Для гри в п'ятнадцять станів задачі - це просто деяке конкретне розташування фішок. Початкова і цільова конфігурації являють собою відповідно початковий і цільовий стани. *Простір* станів, досяжних з початкового стану, складається з усіх тих конфігурацій фішок, що можуть бути освічені в результаті припустимих правилами переміщень фішок. Багато задач мають надзвичайно великі (якщо не нескінченні) простори станів.

Оператор перетворить один стан в інше. Гру в п'ятнадцять природно усього інтерпретувати як гру, що має чотири оператори, що відповідають наступним позиціям: пересунути порожню клітку (пробіл) уліво, нагору, вправо, униз. У деяких випадках оператор може виявитися незастосовним до якого-то стану. Мовою станів і операторів рішення деякої проблеми є послідовність операторів, що перетворить початковий стан у цільовий.

Простір станів, досяжних з даного початкового стану, корисно уявляти собі у виді графа, вершини якого відповідають цим станам. Вершини такого графа зв'язані між собою дугами, що відповідають операторам.

Оператори приводять один стан в інший. Таким чином, їх можна розглядати як функції, визначені на безлічі станів і приймаючі значення з цієї безлічі. Тому що процеси рішення задач засновані на роботі з описом станів, то будемо припускати, що оператори - функцій цих описів, а їхнього значення - нові описи. У загальному випадку будемо припускати, що оператори - це *обчислення*, що перетворять одні описи станів в інші.

У деяких задачах оптимізації недостатньо знайти будь-який шлях, що веде до мети, а необхідно знайти шлях, оптимізуючи деякий критерій (наприклад, мінімізуючи число застосувань операторів). З такими задачами найпростіше працювати, зробивши так, щоб пошук не закінчувався доти, поки не буде знайдене деяке оптимальне рішення.

Таким чином, для повного представлення задачі в просторі станів необхідно задати:

а) форму опису станів і, зокрема, опис початкового стану;

б) безліч операторів і їхніх впливів на описи станів;

в) властивості опису цільового стану.

Простір станів корисно уявляти собі у виді спрямованого графа (Малюнок 4.2).

Граф складається з безлічі (не обов'язково кінцевих) *вершин*. Деякі пари вершин з'єднані за допомогою *дуг*, і ці дуги спрямовані від одного члена цієї пари до іншого. Такі графи звуться *спрямованими графами.* Якщо деяка дуга спрямована від вершини n_i до вершини n_j, то говорять, що вершина n_j є дочірньою для вершини n_i, а вершина n_i є батьківською вершиною для n_j. Може виявитися, що наші дві вершини будуть дочірніми друг для друга; у цьому випадку пари спрямованих дуг називається іноді ребром графа. У випадку, коли граф використовується для представлення простору станів, з його вершинами зв'язують опис станів, а з його дугами - оператори.

Послідовність вершин $n_{i1},n_{i2},...,n_{ik.}$, у якій кожна вершина n_{ij} дочірня для $n_{i,j-1}$, $j=2,k$, називається шляхом довжини k від вершини n_{i1}, до вершини n_{ik}. Якщо існує шлях, що веде від вершини n_i до вершини n_j, то вершину n_j називають досяжною з вершини n_i або нащадком вершини n_i . У цьому випадку вершина n_i називається також предком для вершини $n_{j.}$ Видно, що проблема перебування послідовності операторів, що перетворять один стан в інше, еквівалентна задачі пошуку шляху на графі, що і є пошук методом редукції.

4.3. Процеси пошуку на графі

Граф визначається як безліч вершин разом з безліччю ребер, причому кожне ребро задається парою *вершин*. Якщо ребра спрямовані, то їх також називають *дугами*. Дуги задаються *упорядкованими* парами. Такі графи називаються *спрямованими*. Ребрам можна приписувати вартості, імена або мітки довільного виду, у залежності від конкретного додатка.

При формулюванні задачі рішення виходить у результаті застосування операторів до описів станів доти, поки не буде отримане вираження, що описує стан, що відповідає досягненню мети. Усі методи перебору, що ми будемо обговорювати, можуть бути змодельовані за допомогою наступного теоретико - графового процесу:

Початкова вершина відповідає описові початкового стану. Вершини, що безпосередньо виходять з даної, виходять у результаті використання операторів, що застосовні до опису стану. Нехай Г- деякий спеціальний оператор, що будує *усі* вершини, що безпосередньо виходять з даної. Процес застосування оператора Г до вершини називається *розкриттям* вершини.

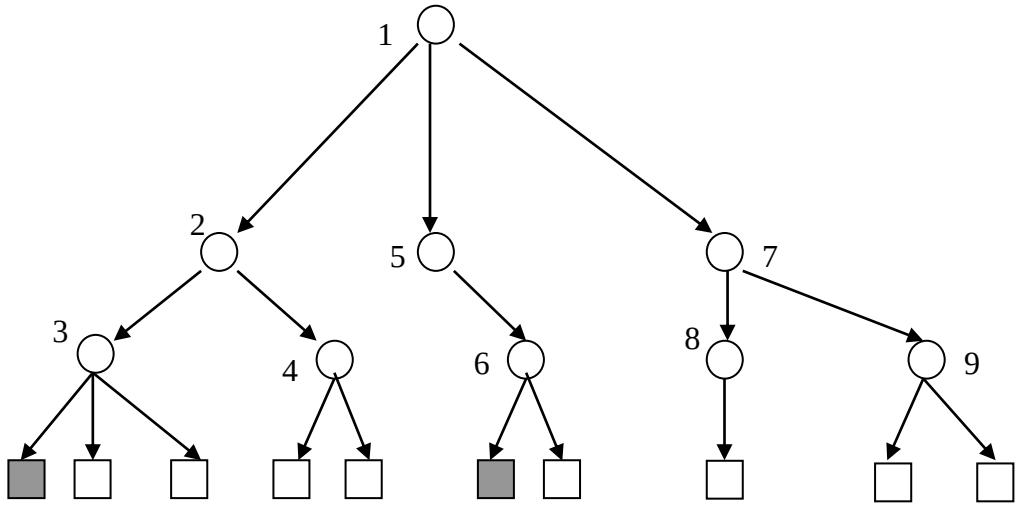
Від кожної такої наступної вершини до породившої її йдуть *покажчики*. Ці покажчики дозволяють знайти шлях назад до початкової вершини, уже після того як виявлена цільова вершина.

Для вершин, що виходять з даної, робиться перевірка, чи не є вони цільовими вершинами. Якщо цільова вершина ще не знайдена, то продовжується процес розкриття вершин (і установки покажчиків). Коли ж цільова вершина знайдена, ці покажчики проглядаються в зворотному напрямку - від цілі до початку, у результаті чого виявляється шлях рішення. Тоді оператори над описами станів, зв'язані з дугами цього шляху, утворять вирішальну послідовність.

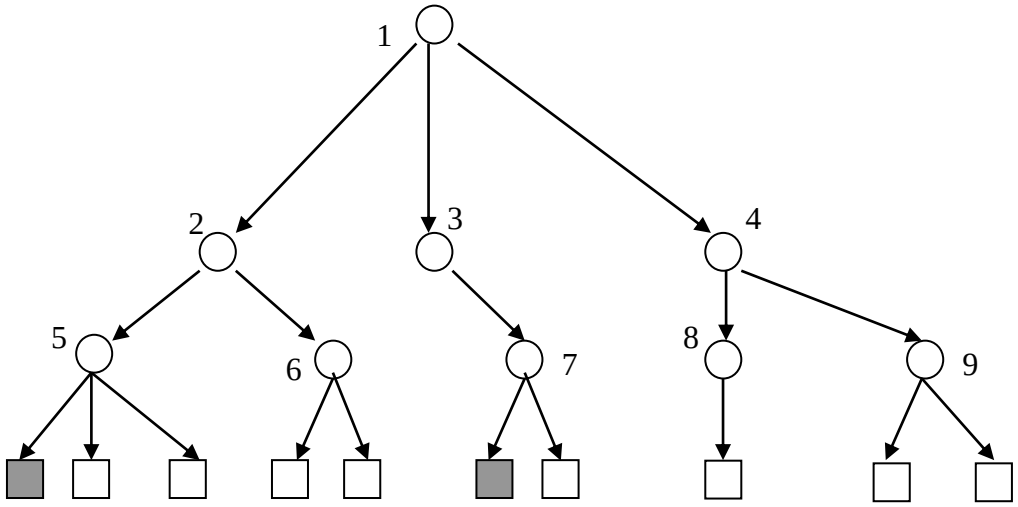
Етапи, зазначені вище, описують основні елементи процесу перебору. При повному описі процесу перебору потрібно ще задати порядок, у якому варто розкривати вершини. Якщо вершини розкриваються в тім же порядку, у якому вони породжуються, то виходить процес, що називається *повним перебором*,

Якщо ж спочатку розкривається завжди та вершина, що була побудована самою останньої, то виходить процес *перебору в глибину*. Можливо, однак, що мається деяка евристична інформація про глобальний характер графа і загальне розташування цілі пошуку. Такого роду інформація часто може бути використана для того, щоб “підштовхнути” пошук убік цілі, розкриваючи в першу чергу найбільш перспективні вершини.

При пошуку методом редукції рішення задачі зводиться до рішення сукупності утворюючих її підзадач. Цей процес повторюється для кожної підзадачі доти, поки кожна з отриманого набору підзадач, що утворять рішення вихідної задачі, не буде мати очевидне рішення. Підзадача вважається очевидною, якщо її рішення загальновідоме, або отримано раніше. Процес рішення задачі розбивка на підзадачі можна представити у виді спеціального спрямованого графа G, називаного І/АБО- графом. У графі виділяються два типи вершин: кон’юнктивні вершини і диз'юнктивні вершини. Графічне представлення вершин приведене далі на Малюнок 6.6.



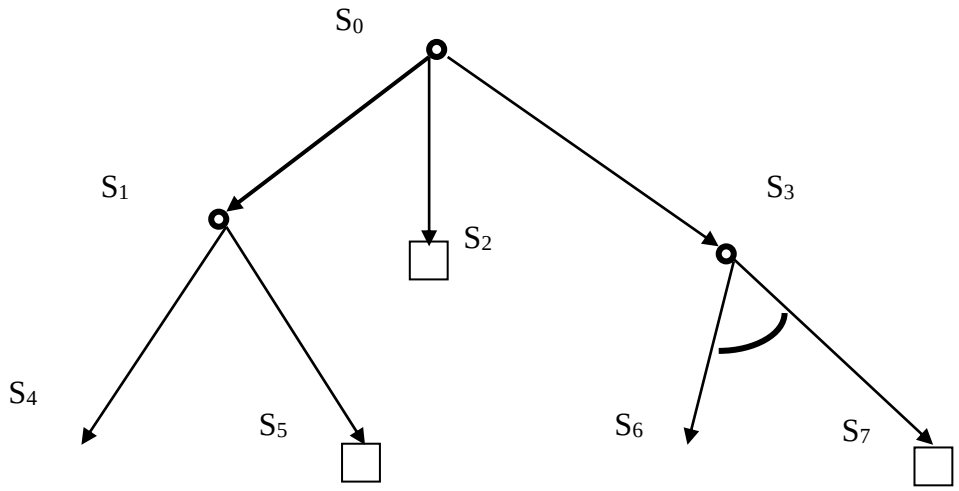
а



б

Малюнок 4.2 Простір станів, побудований пошуком у глибину (а) і пошуком у ширину (б)

Приклад графічного представлення розбивки на підзадачі, приведений на Малюнок 4.3..



Малюнок 4.3.Графічне представлення розбивки на підзадачі

Ціль процесу пошуку в І/АБО графі - показати, що початкова вершина розв'язна, тобто для цієї вершини існує вирішальний граф. Визначення розв'язної вершини в І/АБО графі можна сформулювати рекурсивно в такий спосіб:

- Кінцеві (цільові) вершини розв'язні, тому що їхнє рішення відоме по вихідному припущенню.
- Вершина АБО розв'язна тоді, і тільки тоді, коли розв'язна принаймні одна з її дочірніх вершин.
- Вершина І розв'язна тоді і тільки тоді, коли розв'язна кожна з її вершин.

Вирішальний граф визначається як підграф з розв'язних вершин, кожний показує, що початкова вершина розв'язна.

Наступна програма ілюструє стратегію пошуку в глибину і зворотний ланцюжок міркувань. Припустимо, дерево (Малюнок 4.4) відбиває транспортне повідомлення між населеними пунктами.

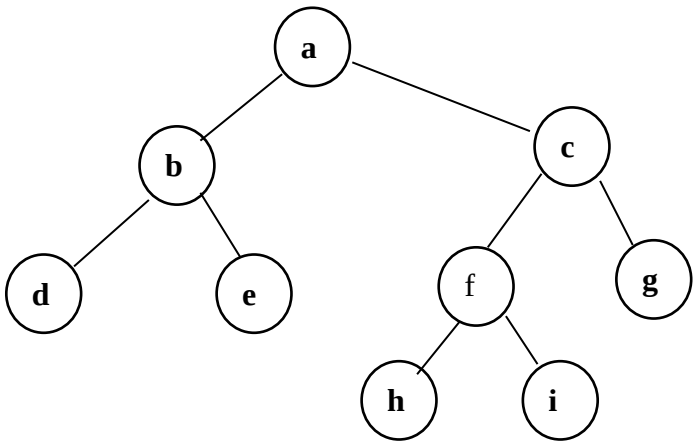
Факти, що означають прямий зв'язок між вузлами графа (містами), можуть виглядати так:

dconnect(a,b) dconnect(c,f)

dconnect(a,c) dconnect(c,g)

dconnect(b,d) dconnect(f,h)

dconnect(b,e) dconnect(f,i)



Малюнок 4.4 Транспортне повідомлення між населеними пунктами

Додамо непрямий зв'язок connect(X,Y) для виключення зациклення.
Правило мовою PROLOG, що визначає маршрут униз по дереву, виглядає так:

connect(X,Y) if dconnect(X,Y).

Факт connect(X,Y)- щирий, якщо істина факт dconnect(X,Y).
Правило, що перевіряє, чи є зв'язок між пунктами X і Y через проміжний пункт Z:

connect(X,Y) if dconnect(X,Z), dconnect(Z,Y).

Просте правило завжди ставлять першим - це зменшить час пошуку рішення. Стратегію пошуку в глибину застосовують для рішення проблеми, де всі шляхи пошуку від вершини дерева до його підстави мають однакову довжину. При неоднаковій довжині придатний пошук у ширину. Текст програми, що перевіряє чи є зв'язок між пунктами X і Y приведен в Додатку 1.

Приведена в попередній главі керуюча структура відноситься до однієї з простих; вона зводиться до перебору правил, поки одне з них не виконається, і повторенню циклу із самого початку. У більшості реальних систем задача керування значно складніше.

4.4. Евристичний пошук

Підхід "*пошук у просторі станів"* сформувався в результаті спроб автоматизації ігор. У більшості ігор мається кінцеве число позицій (або "станів"), що можуть бути досягнуті при використанні фіксованого набору правил, що визначають гру. Тому (принаймні, у теорії) існує можливість опису послідовності гри шляхом перерахування всіх позицій, що можуть бути досягнуті з заданого початкового стану. Ця безліч позицій може бути представлена у виді дерева, що простирається від кореня, що відповідає початковому станові гри, до заключних позицій, що досягається наприкінці гри.

Приведемо всі можливі позиції (Малюнок 4.5)для чотирьох перших ходів у простій грі "хрестики-нулики".

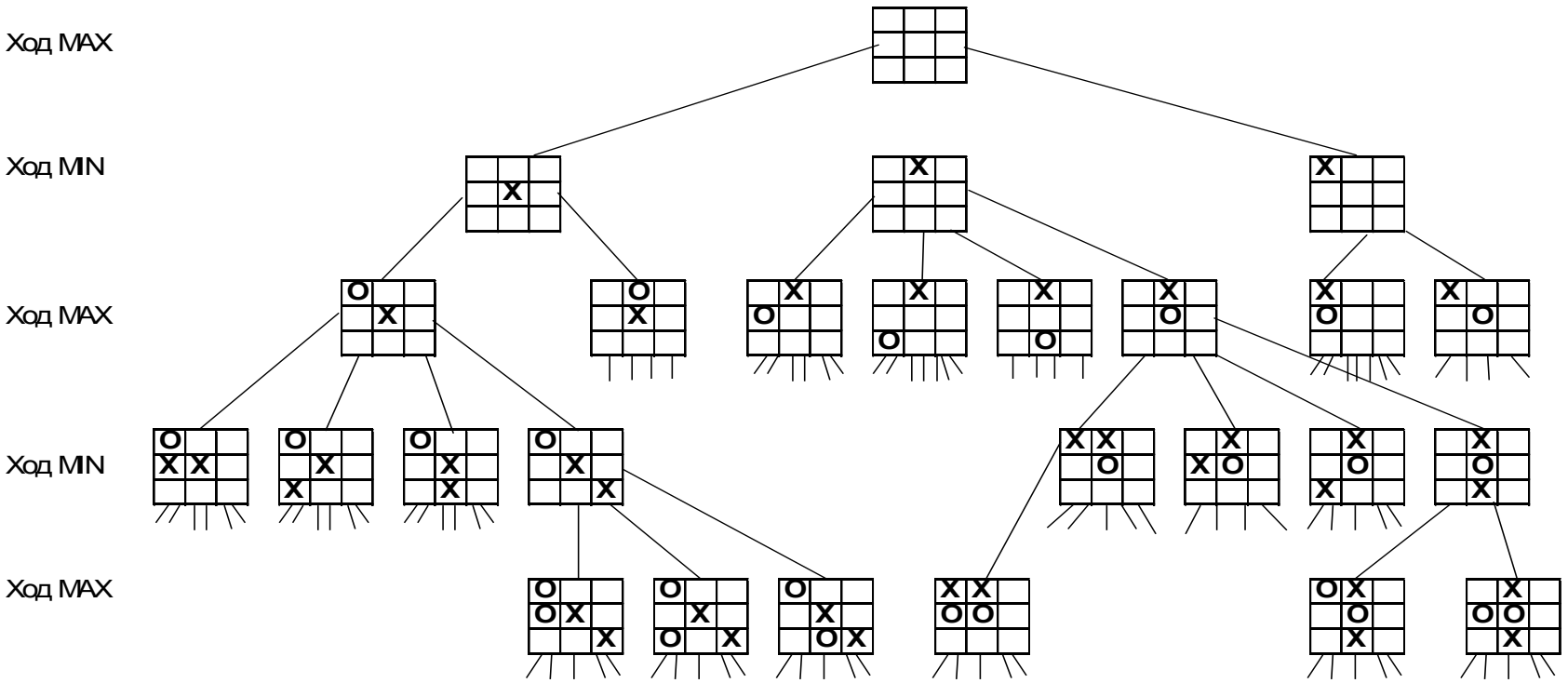
У рамках описаного вище представлення "простору станів" положення фішок (стану), що відповідають успішній грі, можна розглянути серед заключних позицій дерева. Виграшні послідовності ходів можуть бути виявлені в результаті прокладки шляху від вихідного стану до виграшного.

У грі "хрестики-нулики" існує тільки один тривіальний оператор — установка фішки (0 або X) у порожній квадрат матриці. Однак в інших іграх операторів може бути кілька. Наприклад, у шахах запропоновані для кожної з фігур ходи можна розглядати як оператори, що

перетворять картину шахової позиції й у такий спосіб просуваючі гру від одного стану до іншого.

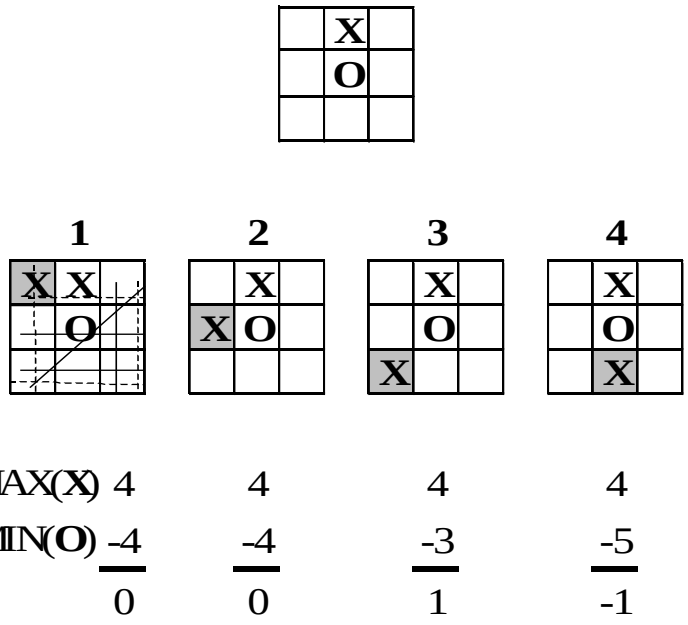
Коли такі діаграми застосовуються в системах, побудованих на основі знань, їх звичайно зображують по-іншому: ціль розташовують нагорі, а вихідну позицію — унизу. Наприклад, при зворотному напрямку пошуку рух походить від мети до початкового стану, а при прямому — від початкового стану до мети.

З досягненням мети за допомогою повного перебору, у просторі станів зв'язана досить серйозна проблема. Навіть у такій простій грі, як “хрестики-нулики”, існує 362880 (9!) можливих позицій. Завдяки симетрії деяких з них цю цифру можна скоротити приблизно до 60 000. У таких складних іграх, як шахи, було б неможливо зробити пошук по всьому дереву гри, тому що перегляд уперед на 5 ходів дає квадрильйон (10^{15}) комбінацій, а 40 ходів (у середньому за гру) дають 10^{120} комбінацій. При цьому з моменту зародження Всесвіт пройшло менш 10^{80} секунд! На кожнім кроці кількість варіантів вибору множить загальне число комбінацій. Цей процес, що одержав назву “комбінаторного вибуху”, виключає застосування стратегії пошуку повним перебором для більшості реальних ігор, а також і для більшості систем, на основі знань. Жоден гравець не проводить пошук чергового ходу простим перебором, а застосовує визначені стратегічні правила для виявлення того, які з ходів забезпечують найбільшу можливість успіху. У багатьох випадках це приводить до значного скорочення числа позицій, що переглядаються. У більшості ігор (і взагалі ситуацій при обробці знань) часто неможливо знайти правила, що гарантують успіх. Разом з тим нерідко вдається установити правила, що забезпечують збільшення імовірності успіху. Такі правила називають евристичними, а пошук, при проведенні якого вони застосовуються, називається *евристичним пошуком*.



Малюнок 4.5 Чотири ходи в грі “хрестики- нулики”

Для евристичного правила потрібна інформація про його ефективність. Ця інформація визначається по деякій “оцінній функції”. У грі “хрестики-нулики” використовується проста оцінна функція. Деяке значення (число) привласнюється кожному наступному можливому ходові. Чим воно вище, тим краще для одного гравця і гірше для іншого.



Малюнок 4.6 Застосування евристичної оцінної функції

Ця процедура називається MINIMAX. Проілюструємо застосування процедури MINIMAX при грі в “хрестики-нулики”

Допустимо, що є два гравці - MAX (граючий фішками X) і MIN (граючий фішками 0). Значення оцінки для доступних їм ходів можуть бути отримані шляхом підрахунку всіх ліній, відкритих для кожного з гравців, а потім вирахування одного числа з іншого.

Стосовно до позиції (Малюнок 4.6), MAX повинний зробити вибір серед ходів 1, 2, 3 і 4. Вони відповідно одержують оцінки 0, 0, 1, -1. MAX вибирає хід 3, тому що він одержав найвищу оцінку для цього раунду.

У грі в “хрестики-нулики” прийнятна конструкція єдиної оцінної функції, що надійно прокладає шлях до оптимального рішення.

Інша стратегія [30] застосовується в грі “Вісімки” (Малюнок 4.7).

Початковий стан

Представлення у вигляді малюнка

4		6
7	2	1
3	8	6

Представлення у вигляді списка 4, 0, 5, 7, 2, 1, 3, 8, 6

Кінцевий стан

Представлення у вигляді малюнка

3	4	8
7	1	5
	2	6

Представлення у вигляді списка 3, 4, 8, 7, 1, 5, 0, 2, 6

Малюнок 4.7 Задача «Вісімки»

Припустимо, що квадратні фішки з номерами переміщаються в квадратній рамці. Задано початковий і кінцевий стани. Задача полягає в перебуванні послідовності переміщень фішок з початкового положення в кінцеве. Будь-яку фішку можна пересунути по горизонталі або вертикалі, якщо поруч є вільне місце, але рухатися по діагоналі заборонено.

Кількість станів занадто велика для їхнього безпосереднього перебору. Замість цього представте програму, що переміщає фішки у визначеному порядку від стану до стану. Обчислюючи евристичну функцію для кожного стану, вона оцінює якість ходів. Таке дослідження може бути виконано декількома способами з використанням евристичних функцій.

Замість сліпого переміщення від одного стану до іншого потрібно проранжувати усі можливі ходи і вибрати той, котрий вважаємо кращим. Звернемося до однієї корисної евристичної функції, називаною *відстанню Хемінга*. Це відстань між даним станом і нульовим. У нашому випадку вона виміряється

числом фішок, що знаходяться не на своїх місцях (порожня позиція не зараховується). Наприклад, у наступній ситуації:

[4,0,5,7,2,1,3,8,6] положення в задачі

[3,4,8,7,1,5,0,2,6] положення цілі

відстань Хемінга дорівнює семи. Якщо процедура пошуку зможе знайти положення, де відстань Хемінга дорівнює нулеві, задача буде вирішена.

Одна зі стратегій, полягає в тім, що за допомогою оцінної функції ви вибираєте найкращий хід і перевіряєте тим же способом (використовуючи оцінну функцію) усі його наслідки, перш ніж почнете які-небудь інші кроки. У випадку тупика потрібно повернутися до найближчої розвилки і спробувати зробити наступний найкращий хід. Ця стратегія відома як *метод ярів*, що є варіантом процедури пошуку в глибину: спочатку перевіряються всі наслідки одного вибору на всю можливу глибину, і тільки потім відбувається черговий вибір.

Метод ярів, хоча і цікавий, мало допоможе при рішенні задачі "Вісімка". Серйозний недолік цієї стратегії полягає в її "короткозорості" - для здійснення вибору використовується тільки локальна інформація.

Існує інший варіант евристичного пошуку, що поєднує в собі позитивні риси пошуку в глибину й у ширину. Він відомий за назвою *Кращий перший крок,* і в цілому дає гарні результати. На кожній стадії пошуку досліджується деяка кількість шляхів, і оцінна функція обчислюється для кінцевого стану цих шляхів. У Кращому першому кроці пошук продовжується від кінцевого досягнутого стану, що має найкращу оцінну функцію нагору до цього місця. У цій стратегії часто відбуваються стрибки від кінця одного шляху до іншого, намагаючись завжди працювати з тим, що є найбільш обіцяним. Для такого пошуку потрібне збереження великої інформації, тому що всі часткові шляхи повинні постійно підтримуватися і багато часу на обчислення.

Є ще одна оцінна функція, досить проста, але потребуюча значно більше часу для обчислень. Вона обчислює для кожної фішки відстань від її вихідного положення до кінцевого, а потім підсумовує ці відстані. Останні обчислюються за визначеними правилами. Якщо фішка переміщається уздовж верхньої рамки, то

число кроків дорівнює числу фішок, що торкаються, плюс один. У тому випадку, коли фішка переміщається від центра до периферії (або навпаки), число кроків дорівнює одиниці, якщо вона знаходиться в позиції, що примикає, і дорівнює двом, - якщо в діагональній позиції.

На жаль, евристичний пошук може лише прискорити рішення задачі "Вісімка". Немає пошукового методу, що швидко веде до прямої відповіді. Це більш складна проблема, чим могло показатися спочатку, у всякому разі, якщо застосовується рішення, засноване на пошуку. Очевидно, люди вирішують такі задачі по-іншому.

Існують деякі більш "хитрі" стратегії, що стосовно до конкретних проблемних областей дають кращі результати, чим досліджені типи пошуку. Огляд подібних стратегій даний у підручнику Elaine Rich, Artificial Intelligence, McGraw-Hill, New York, 1983.

Однак часто дуже важко, якщо не неможливо, знайти таку функцію для багатьох задач реального світу. Одна зі складностей, про яку згадувалося вище стосовно до шахів, полягає в тім, що в який-небудь з позицій гри, можливо, варто зробити хід, що здається менш удалим, для того, щоб пізніше досягти переваги. Багато зусиль у роботі, зв'язаної з керуючими структурами в області штучного інтелекту, було витрачено при спробах справитися з подібними складними ситуаціями. Дві універсальні стратегії, застосовані в експертних системах, пропонують наступне:

а) організувати простір задачі таким чином, щоб на початку пошуку рішення можна було сказати, наскільки воно буде успішним, у цьому випадку може бути виключена необхідність перебору всієї безлічі шляхів пошуку;

б) побудувати простір задачі у виді ряду підпросторів з мінімальними взаємозв'язками або без взаємозв'язків між ними. У результаті цього кожна з підзадач може бути вирішена незалежно від рішення наступних задач.

У наступній програмі мовою Prolog визначається оптимальний маршрут за заданими критеріями: по мінімальній витраті бензину і по мінімальній відстані.

Дані про відстані між пунктами знаходяться у файлі pal.ddd у виді:

```
sosed("pc","a",1,100).  sosed("a","b",1,120).  sosed("b","c",2,400).
```

```
sosed("c","d",3,308).  sosed("a","d",2,205).  sosed("a","e",3,320).
sosed("e","h",3,290).
```

Предикат dcon перевіряє, чи є пункти сусідніми:

```
dcon(X,Y,M,N) :- sosed(X,Y,M,N);sosed(Y,X,M,N).
```

За допомогою правила summa підсумовуємо відстань між поточними сусідніми пунктами і витрату бензину:

```
summa([_],N,S,_,_) :- S=N.
summa([X,Y|L],N,S,Км,Бензин) :-
        bound(Км),dcon(X,Y,M,_),
         K=M+N, summa([Y|L],K,S,Км,Бензин);
        bound(Бензин), dcon(X,Y,_,M), K=M+N,
        summa([Y|L],K,S,Км,Бензин).
```

Формуємо списки з усіма можливими варіантами шляху і використовуючи правило min визначаємо шлях з мінімальним значенням:

```
min([],Tmp,Cnt,_,Min,N) :- Min=Tmp, N=Cnt.
min([X|L],Tmp,Cnt,K,Min,N) :- Tmp>X, C=K, KK=K+1, MM=X,
                    min(L,MM,C,KK,Min,N);
                    KK=K+1, min(L,Tmp,Cnt,KK,Min,N).
```

Повний текст з інтерфейсом користувача приведений у Додатку 2.

4.5. Експертна система на правилах

У системі, що базується на правилах, результат є дією одного з продукційних правил. Ці продукційні правила визначаються вхідними даними.

Таким чином, експертна система, що базується на правилах (на Пролозі) містить безліч правил, що викликаються за допомогою вхідних даних у момент

зіставлення. Експертна система також містить інтерпретатор у механізмі висновку, що вибирає й активізує різні модулі системи.

Роботу цього інтерпретатора можна описати послідовністю з трьох кроків:

1. Інтерпретатор зіставляє зразок правила з елементами даних у базі знань.

2. Якщо можна викликати більш одного правила, то інтерпретатор використовує механізм дозволу конфлікту для вибору правила.

3. Інтерпретатор застосовує обране правило, щоб знайти відповідь на питання.

Цей трьох кроковий процес інтерпретації є циклічним і називається циклом "розпізнавання-дія".

У системі, що базується на правилах, кількість продукційних правил визначає розмір бази знань. Деякі найбільш складні системи мають бази знань з більш ніж 5000 продукційних правил. Ви можете почати з невеликої кількості правил і додавати їх у базу знань у міру розширення експертної системи.

Може бути більш важливим, чим розміри бази знань, є структура самих продукційних правил. От кілька рекомендацій по побудові сумісних правил:

1. Використовувати мінімально достатня безліч умов при визначенні продукционного правила.

2. Уникати суперечних продукційних правил.

3. Конструювати правила, спираючи на структуру властивої предметної області.

Експертна система на Турбо-Пролозі - система для ідентифікації породи собак. Припустимо, що користувач повідомив безліч характеристик собаки у відповідь на питання експертної системи.

Інтерпретатор працює в циклі розпізнавання-дія. Якщо характеристики порівнянні з характеристиками породи собаки, що складають частину бази знань, тоді викликається відповідне продукційне правило й у результаті ідентифікується порода. Потім результат повідомляється користувачу. Аналогічно, якщо порода не ідентифікована, це теж повідомляється користувачу.

Тепер розглянемо дві характеристики породи собак, що міститися в базі знань. Гонча має коротку вовну, зріст менше 22 дюймів, довгі вуха і гарний характер. Дог

має коротку вовну, звисаючий хвіст, довгі вуха, гарний характер і вагу більш 100 фунтів.

З цього опису ясно, що обидві породи мають коротку вовну, довгі вуха і гарний характер. Зріст гончої менше 22 дюймів у той час як нічого не сказано про зріст дога. Дог має звисаючий хвіст і вагу більш 100 фунтів – характеристики відсутні для гончої. Опису двох собак у термінах зазначених характеристик досить, щоб розрізнити ці дві породи, і навіть відрізнити їх від будь-якої іншої породи в базі знань.

Наступні продукційні правила можуть бути складені по зазначених характеристиках:

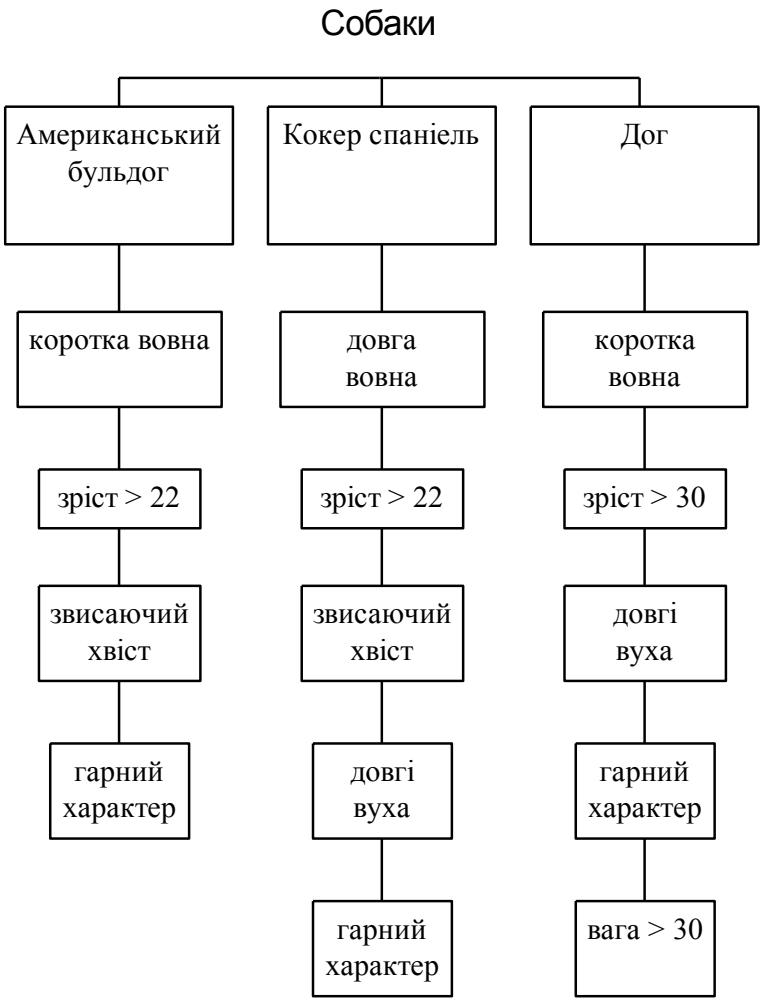
```
dog_is("Great Dane") :-
        it_is("short-haired dog"),
        positive(has,"low-set tail"),
        positive(has,"good natured personality"),
        positive(has,"weight over 100 lb"),!.
dog_is("Beagle"):-
        it_is("short-haired dog"),
        positive(has,"height under 22 inches"),
        positive(has,"long ears"),
        positive(has,"good natured personality"), !.
```

У попереднім правилі довжина вовни може бути представлена за допомогою предиката positive у виді:

```
positive(has,"short-hair").
```

Але використання предиката it_is дозволяє обмежити "простір пошуку" (кількість даних, що перевіряються при пошуку рішення) одним піддеревом структури, що містить інформацію про породи собак (Малюнок 4.8).

Експертна система, що базується на правилах, дозволяє проектувальнику будувати правила, що природним образом поєднують у групи зв'язані фрагменти знань. Кожне продукційне правило може бути незалежним від інших. Ця незалежність робить базу продукційних правил семантично модульної, тобто групи інформації не впливають один на одного. Більше того, модульність бази правил дозволяє розвивати базу знань, збільшуючи її.



Малюнок 4.8. Дерево, що містить інформацію про різні породи собак

У Пролозі продукційні правила містяться у програмі, і, отже, розміри програми збільшуються по мірі додавання правил. Розміри пам'яті зрештою обмежують число правил. У цьому випадку використання системи на правилах стає проблематичним. У той же час, у системі, що базується на логіці, де база знань знаходиться у файлі на диску, обмеження на розміри бази знань не накладаються. Тому система, заснована на логіці, переважніша в цьому випадку.

Відомо також, що в медичних експертних системах знайшла застосування організація знань у виді статичних описів хвороб, визначених через відповідні симптоми. Подібні програми розглянуті в книзі Е. І. Єфімова “Решатели интеллектуальных задач”, Москва, Наука, 1982. Задача діагностичних програм,

зводиться до пошуку найменьшого набору хвороб, здатних бути причиною заданого сполучення симптомів (система INTERNIST).

Текст програми експертизи по породах собак реалізуючої продукційну систему, що базується на правилах приведений у Додатку 3.

4.6. Експертні системи, що базуються на логіці

В експертних системах, що базуються на логіці, база знань складається з тверджень у виді пропозицій логіки предикатів.

Такі пропозиції можуть групуватися, утворювати базу даних Прологу. Правила можуть або описувати дані, або керувати процесом внутрішньої уніфікації Прологу. Так само як і в системі на правилах експертна система, що базується на логіці, має безліч правил, що можуть викликатися за допомогою даних із вхідного потоку. Система має також інтерпретатор, що може вибирати й активізувати модулі, що включаються в роботу системи.

Інтерпретатор виконує різні функції усередині системи на основі наступної схеми:

1. Система має пропозиції в базі знань, що керують пошуком і зіставленням. Інтерпретатор зіставляє ці пропозиції з елементами даних у базі даних.

2. Якщо може бути викликане більш одного правила, то система використовує можливості Прологу для роз’язування конфлікту. Отже, програмісту не потрібно розглядати потенційно можливі конфлікти.

3. Система одержує результати унифікаційного процесу автоматично, тому вони можуть направлятися на потрібний пристрій висновку інформації.

Так само як і в системі, що базується на правилах, даний циклічний процес є процесом розпізнавання-дії. Краса і великі можливості системи, заснованої на логіці, полягають у тому, що вона відбиває структуру самого Прологу. Під цим розуміється той факт, що система дуже ефективна в роботі.

Найбільш важливим аспектом для бази знань у системі, заснованої на логіці, є проектування бази знань, її тверджень і їхньої структури. База знань повинна мати недвозначну логічну організацію, і вона повинна містити мінімум надлишкової

інформації. Так само як і в системі, що базується на правилах, мінімально достатня кількість даних утворює найбільш ефективну систему. Твердження бази знань для гончої і дога виглядають так:

```
rule(1,"dog","Beagle",[1,2,3,4]).
rule(2,"dog","Great Dane",[1,5,3,4,6]).
cond(1,"short-haired").
cond(2,"height under 22 inches").
cond(3,"longer ears").
cond(4,"good natured personality").
cond(5,"low-set tail").
cond(6,"weight over 100 lb").
```

У кожній пропозиції типу rule перший аргумент - номер правила, другий аргумент - тип об'єкта "dog" і третій аргумент - порода собаки. У нашому випадку це гонча чи дог. Список цілих чисел задає номера умов із пропозицій типу cond (умова). Пропозиції типу cond містять усі характеристики породи, представленої в базі знань.

Списки номерів умов служать для збереження безлічі фактів, згідно яким вибираються пропозиції типу rule. Інтерпретатор в експертній системі, що базується на логіці, використовує ці номери умов, щоб робити відповідний вибір.

Додавання і відновлення пропозицій бази знань є простими операціями retract і assert. Експертні системи, що базуються на логіці, легко проектувати, розвивати і підтримувати в Турбо-Пролозі, тому що по мірі розширення бази знань програма не вимагає модифікації. Розширення, насамперед, полягає в поступовому додаванні нових тверджень.

Класифікація в базі знань може ґрунтуватися на деревоподібній структурі (Малюнок 4.9). Відповідно до цієї деревоподібної класифікації породи розділені на короткововневі і довгововневі. Англійський бульдог, гонча, дог і американський фокстер'єр є короткововневим , а коккер спанієль, ірландський сеттер, колі і сенбернар - довгововневі.

Малюнок 4.9. Деревоподібна структура бази знань ЭС для вибору породи собаки

Для ідентифікації породи усередині кожної підмножини можна використовувати список атрибутів. Кількість характеристик буде визначати ступінь точності класифікації. Відмінной не обов'язково є яка-небудь єдина характеристика - уся безліч атрибутів використовується для досягнення цілей у споруджуваних правилах. З восьми перерахованих нижче, три є необхідними, тому що жоден з них не характерний для всіх порід одночасно. Атрибут, що характеризує всі об'єкти одночасно, можливо, не є необхідним у безлічі даних. У нашій експертній системі використовуються наступні атрибути:

1. коротка вовна;
2. довга вовна;
3. зріст менше 22 дюймів;
4. зріст менше 30 дюймів;
5. низько звисаючий хвіст;
6. довгі вуха;
7. гарний характер;
8. вага більше 100 фунтів.

Для кожної породи справедливі наступні характеристики:

Порода	Характеристики
Англійський бульдог	1,3,5,7
Гонча	1,3,6,7
Дог	1,4,6,7,8
Американська гонча	1,5,6,7,
Коккер спанієль	2,3,5,6,7
Ірландський сеттер	2,4,6
Колі	2,4,5,7
Сенбернар	2,5,7,8,

У нашому випадку при проектуванні бази знань деревоподібна структура, безліч ідентифікуючих характеристик і набори номерів характеристик для кожної породи складають робочу модель бази знань для вибору породи.

Текст програми експертизи по породах собак, що базується на логіці приведений у додатку 4.

Спочатку введення користувачем слова dog приводить до того, що змінна Mygoal (моя ціль) одержує значення "dog" (собака).

Застосовується твердження бази знань rule(1,"dog","short-haireddog",[1]) і облікова перемінна COND одержує значення 1.

Далі правило rule передає цей параметр правилу check (перевірка). У свою чергу правило check здійснює доступ до правила cond бази знань з параметром BNO рівним 1. Правило check передає це значення предикату fronttoken, щоб створити значення _COND. Це правило дає невдачу.

Правило check повертається до правила cond і COND містить значення параметра. Потім правило check здійснює доступ до значення "short-haired" (короткововневий) і передає його у змінну TEXT правилу inpq. Правило inpq видає на екран текстовий рядок:

Question:- short-haired? (Питання:- короткововневий?)

Користувачу повідомляється, що він повинний натиснути 1 для позитивної відповіді і 2 для негативного. Правило inpq приймає відповідь користувача в нашому прикладі це 2, і інтерпретує його як негативну.

Процес продовжується з наступним твердженням rule бази знань, тобто RNO рівним 2. Тепер правило check використовує значення CON рівне 2 для поточного правила rule, повторює цикл побудови списку значень COND і запитує користувача про додаткове введення.

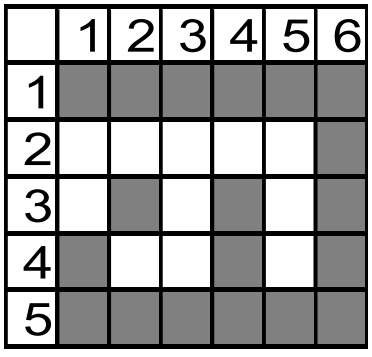
Ґрунтуючись на відповідях користувача, що вводяться, доступ до тверджень rule і cond відбувається в наступним порядку:

rule(1), cond(1),
rule(2), cond(2),
rule(7), cond(3),
rule(7), cond(5),
rule(7), cond(6),
rule(7), cond(7).

Наприкінці цього процесу змінна COND типу список має значення 3,5,6,7. Цей список зіставляється зі списком умов у правилі 7. Зіставлення зв'язується з класифікованим об'єктом "Cocker Spaniel" (коккер спанієль) у твердженні rule, і механізм висновку, таким чином, знайшов необхідний результат.

Розглянемо програму пошуку виходу з лабіринту [39]. Опишемо лабіринт (Малюнок 4.10) за допомогою предиката a:

```
a(1,1,стена).a(1,2,стена).a(1,3,стена).a(1,4,стена).
a(2,1,пусто).a(2,2,пусто).a(2,3,пусто).a(2,4,пусто).
a(3,1,выход).a(3,1,пусто).a(3,2,стена).a(3,3,пусто). a(3,4,стена).
a(4,1,стена).a(4,2,стена).a(4,3,пусто).a(4,4,пусто).
a(5,1,стена).a(5,2,пусто).a(5,3,пусто).a(5,4,стена).
```



Малюнок 4.10 Лабіринт

Місце старту в лабіринті задаємо в меті (клітка 4 2):

путь(a(4,2,пусто),P,[a(4,2,пусто)]), write(P).

P - список позицій, що ведуть до виходу.

Шукаємо шлях з вихідної позиції в північному напрямку. Якщо шляху немає то йдемо на південь. Якщо шляху немає йдемо на захід. Якщо не можна, йдемо на схід. Якщо сусідня позиція є стіною (правило можно_идти), то робимо поворот на 90° (перевірка наступного правила путь). Щоб не ходити колами, ведемо список позицій у який побували (список Были).

Результат роботи програми:

Выход31[a(4,2,"пусто"),a(5,2,"пусто"),a(5,3,"пусто"),a(4,3,"пусто"),
a(3,3,"пусто"),a(2,3,"пусто"),a(2,2,"пусто"),a(2,1,"пусто"),a(3,1,"пусто")]

Описана процедура не обов'язково знаходить короткий шлях до виходу.

Повний текст програми приведений у Додатку 5.

Контрольні питання

1. Викличте експертну систему, що базується на правилах, для вибору породи собаки. Уведіть різні послідовності відповідей "yes" і "no", спостерігаючи, як працює програма.

2. Модифікуйте експертну систему, що базується на правилах, для вибору породи собаки, додавши гіпотетичну породу. Напишіть продукційне правило для цієї породи собаки і включити правило в програму. Ваші характеристики собаки повинні бути комбінацією характеристик вже існуючих у програмі, але комбінація повинна відрізнятися від комбінацій для інших порід.

3. Викличте програму експертної системи для вибору породи, засновану на логіці. Проведіть кілька консультацій і подивиться , як працює система.

4. Модифікуйте програму, включивши інформацію про іншу породу собаки. Напишіть відповідне твердження логіки предикатів, що містить її характеристики у виді списку цілих чисел. Переконайтеся, що кожен список унікальний.

5. Напишіть усі правила трансформації, що описують операції в задачі "Вісімка". Керуйтеся правилом, приведеним у цій главі.

6. Напишіть предикат для обчислення відстані Хемминга між двома списками цілих чисел. Перші два аргументи повинні бути списками, а третій - відстанню.

7. Проаналізуйте гру хрестики-нулики на полі 4x4. Для виграшу потрібно поставити підряд чотири знаки. Яку гарну евристику можна придумати для цієї гри?

5.Системи з дошкою оголошень

В останні роки в розробці архітектури експертних систем з'явився новий напрямок [35], що одержало назву системи з дошкою оголошень (blackboard systems). Системи з такою архітектурою можуть эмулірувати режим побудови, як прямої ланцюжка логічного висновку, так і зворотної, а також поперемінно застосовувати ці режими в процесі роботи. Крім того, застосування систем з дошкою оголошень спонукує інженерів по знаннях до ієрархічної організації і знань щодо предметної області, і простору часткових і повних рішень. Таким чином, ця архітектура дуже добре підходить для рішення задач проектування, для яких характерно велике, але факторизуемий багатомірний простір рішень. Системи з подібною архітектурою застосовують для інтерпретації даних, наприклад, розпізнавання графічних зображень і мови, аналізу і синтезу багатомірних структур протеїнів і планування.

Як недолік, системи з дошкою оголошень вимагають значних обчислювальних ресурсів.

5.1. Принцип організації систем з дошкою оголошень

В основу організації систем цього типу покладена наступна ідея.

Уявіть собі групу експертів, що сидять біля класної дошки (чи великої дошки оголошень) і намагаються вирішити яку-небудь проблему.

Кожен експерт є фахівцем у якійсь визначеній області, що має відношення до рішення проблеми.

Формулювання проблеми і вихідних даних записані на дошці.

Експерти допитливо вдивляються в те, що написано на дошці, і кожний з них думає над тим, чим він може допомогти в рішенні проблеми.

Якщо хто-небудь з експертів почуває, що йому є що сказати з цього приводу, він виконує відповідні обчислення і записує результати усі на тій же дошці.

Цей новий результат може дозволити й іншим експертам внести значний вклад у рішення проблеми.

Процес припиняється (а експерти розходяться по будинках), коли проблема буде вирішена.

Така методика спільного рішення проблем буде ефективна, якщо дотримуються визначеної угоди, а саме:

- всі експерти повинні говорити на той самій мові, хоча при записі результатів на дошці можуть використовуватися і різні схеми позначень;

- повинний існувати якийсь протокол визначення черговості "виступів", що набирає сили в ситуації, коли відразу кілька експертів хапаються за крейду і направляються до дошки.

Якщо розглядати працюючу по такому принципі систему з погляду організації процесу обчислень, то в системі можна виділити наступні структурні компоненти. Знання про предметну область розділені між незалежними *джерелами знань (KS – knowledge sources),* що працюють під керуванням *планувальника (sheduler).* Рішення формується в деякій глобально доступній структурі, яку будемо називати *дошкою оголошень (blackboard).* Таким чином, у цій системі всі знання "як надійти" будуть представлені не у виді єдиного набору правил, а у виді набору програм. Кожний з компонентів цього набору може мати у своєму розпорядженні власний набір правил або сумішшю правил і процедур.

Функції дошки оголошень багато в чому подібні з функціями робочої пам'яті в продукційних системах, але її організаційна структура значно складніше. Як правило, дошка оголошень розділяється на кілька *рівнів* опису, причому кожен рівень відповідає визначеному ступеню деталізації. Дані в межах окремих рівнів дошки оголошень представляють ієрархії чи об'єкти графів, тобто структури більш складні, ніж вектори, що використовувалися в робочій пам'яті продукційних систем. У найсучасніших системах може бути навіть кілька дощок оголошень.

Джерела знань формують об'єкти на дошці оголошень, але це виконується за допомогою планувальника. Звичайно *запису активізації джерел знань* містяться в

спеціальний список вибору, відкіля їх витягає планувальник. Джерела знань спілкуються між собою тільки через дошку оголошень і не можуть безпосередньо передавати дані один одному чи запускати виконання яких-небудь процедур. Тут є визначена аналогія з організацією роботи продукційних систем, у яких правила також не можуть безпосередньо активізувати один одного: усе повинно проходити через робочу пам'ять.

5.2. Система HEARSAY

Архітектура на основі дошки оголошень виросла з розробленої наприкінці 70 років системи розпізнавання мови HEARSAY-II і HEARSAY-III. Програмування комп'ютера з метою розпізнавання мови - це одна з найбільш складних задач на той час, за які коли-небудь бралися фахівці в області штучного інтелекту. Її рішення вимагає:

- складної обробки сигналів;
- зіставлення фізичних характеристик звукових сигналів із символічними елементами природної мови;
- виконання пошуку у великому просторі можливих інтерпретацій, у якому об'єднані ці різні по своїй природі елементи.

Для рішення цієї проблеми була обрана методика, заснована на виділенні декількох рівнів абстракції опису аналізованих даних. Самим нижчим є рівень фізичних акустичних сигналів, на якому формується звуковий спектр аналізованих сигналів. На наступних рівнях інформація проходить через нашарування лінгвістичних абстракцій з усе більш збільшеним: рівнем спільності понять - фонеми, силаби (співзвуччя), морфеми, слова, вирази і пропозиції.

Приступаючи до розробки системи, її творці розуміли, що з кожним рівнем аналізу зв'язана окрема галузь знань - аналіз звукових сигналів фонетика, лексичний аналіз, граматика, семантика, ораторське мистецтво. Жодна галузь по окремості не здатна надати досить інформації для того щоб вирішити проблему. Представимо, наприклад, що, користуючись методами обробки акустичних сигналів, змогли розкласти вихідний звук на фонеми. Але без додаткової інформації все рівно не вдається виділити зміст виразів, подібно наступним: I scream (я викликаю) і ice cream

(морозиво) чи please let us know (будь ласка, дайте нам знати) і please lettuce no (будь ласка, без салату). Таким чином, хоча кожен окремий вид (набір) знань відіграє істотну роль у рішенні проблеми і кожний з них може бути представлений у програмі більш-менш незалежно від інших, автоматичне розпізнавання мови вимагає використання всіх цих знань спільно.

При розпізнаванні мови дослідникам приходиться зіштовхуватися ще з однією проблемою, що також можна віднести до числа ключових, — проблемою невизначеності. Вона виявляється на всіх рівнях представлення інформації:

- дані неповні і зашумлени;
- відсутня однозначна відповідність між даними на сусідніх рівнях: прикладом може служити відповідність між рівнями фонем і лексичних при аналізі виразів I scream і ice cream ;
- важливу роль грають лінгвістичний і значеннєвий контексти; інтерпретація сусідніх елементів робить більш-менш ймовірними різні варіанти інтерпретації поточного сегмента.

Більш традиційні підходи до розпізнавання мови засновані на використанні статистичних моделей з теорії передачі інформації для визначення кореляційного зв'язку між сегментами. Підхід, що базується на знаннях, зажадав істотного перегляду методів обробки невизначеності.

Перелічимо вимоги, яким повинна задовольняти ефективно працююча система розпізнавання мови, заснована на знаннях.

1) З усіх можливих послідовностей операцій (приватних рішень) хоча б одна повинна приводити до коректної інтерпретації.
2) Процедура аналізу наявних варіантів інтерпретації повинна давати коректному варіанту більш високу оцінку, чим іншим конкуруючим варіантам. Іншими словами, правильна інтерпретація з урахуванням вимови повинна бути оцінена вище, ніж інші варіанти інтерпретації, не враховуючих особливостей індивідуальної дикції.
3) Обчислювальні ресурси (пам'ять і час обчислень), необхідні для відшукання

вірної інтерпретації, не повинні перевищувати визначений поріг. Система розпізнавання, що через пару днів видасть результат, нехай і правильний, і зажадає пам'яті обсягом декілька гигабайт, навряд чи кому-небудь буде потрібна.

У приведеному списку перша і третя вимоги у визначеній мирі суперечат один одному. Для того щоб коректне рішення споконвічне було присутнє в просторі гіпотез, на стадії формування гіпотез мимоволі приходиться бути марнотратним, що при великому словнику може привести до комбінаторного вибуху елементів рішень. Вихід може бути знайдений тільки при використанні надзвичайно дотепних евристик. Таким чином, найважливішою передумовою досягнення успіху в створенні такої системи є розробка придатної процедури оцінки варіантів (друге з перерахованих вище вимог).

5.3. Використання джерел знань у системі HEARSAY

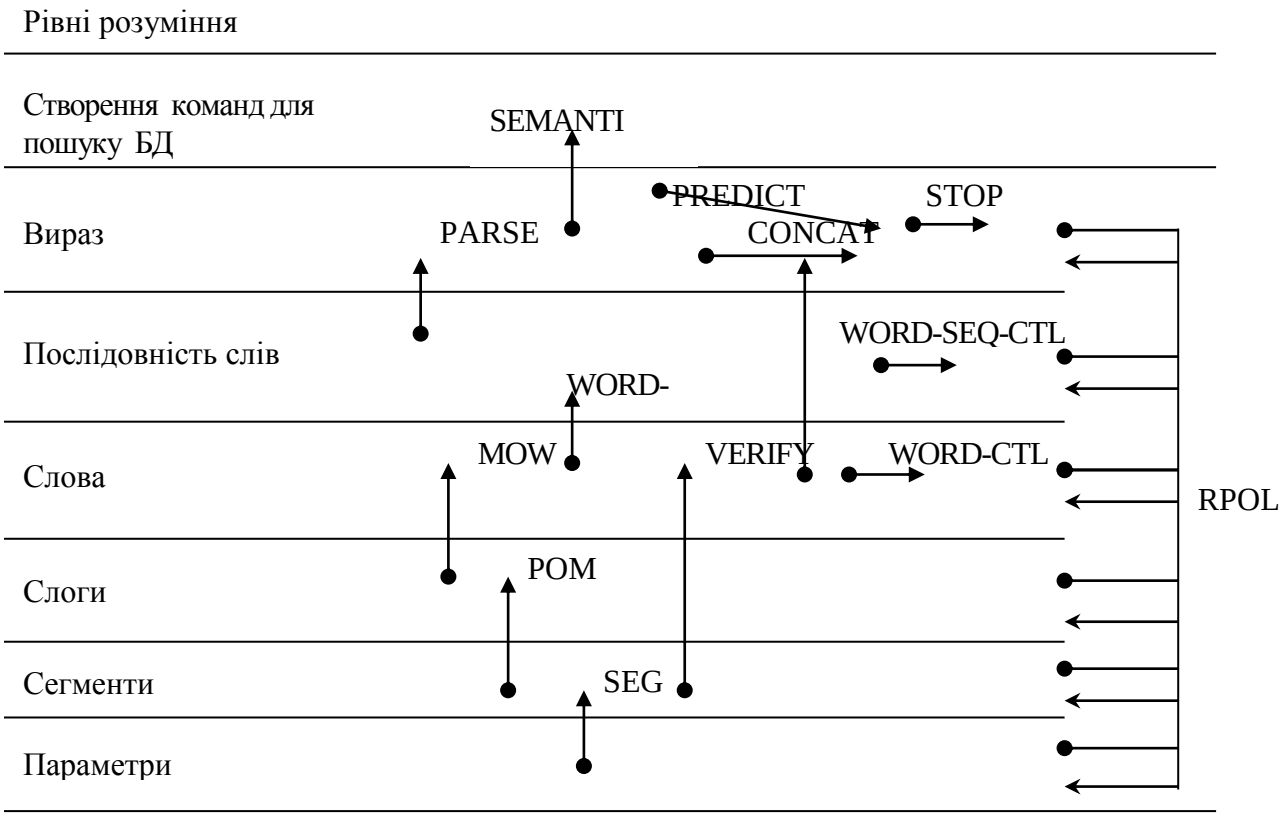
Для генерації, комбінування і розвитку гіпотез інтерпретації в системі HEARSAY-II використовується кілька джерел знань. Створені гіпотези (інтерпретації) різного рівня абстракції зберігаються на дошці оголошень.

Кожне джерело знань можна вважати в першому наближенні набором пара "умова-дія", хоча вони можуть бути реалізовані й у формі, відмінної від правил, що породжують, (наприклад, умови і дії можуть бути в дійсності довільними процедурами). Потік керування в цій системі також відрізняється від потоку керування в продукційних системах. Замість того щоб у кожнім циклі інтерпретатор аналізував виконання умов, специфікованих у джерелах знань, джерела знань загодя повідомляють про активізовані в них умови , сповіщаючи, який вид модифікації даних буде впливати на виконання цих умов. У результаті система керується перериваннями, а цей режим керування значно ефективніше, ніж режим циклічного перегляду стану, що є основним для продукційних експертних систем. Такий режим нагадує використання демонів у фреймових системах, де потік керування регулюється відновленням даних.

Джерела знань зв'язуються з рівнями дошки оголошень у такий спосіб. Умови, специфіковані в джерелі знань, будуть задовольнятися в результаті відновлення даних на визначеному рівні дошки оголошень. Джерело знань також може записувати дані у

визначений рівень, причому не обов'язково в той же, що впливає на виконання умов. Більшість джерел знань у системі HEARSAY-II організовано так, що вони розпізнають дані на визначеному рівні лінгвістичного аналізу, а виконувані ними операції відносяться до наступного по списку рівню. Наприклад, деяке джерело активізується даними на силабічному рівні і формують лексичну гіпотезу на рівні слів.

Має сім рівнів розуміння від акустичних параметрів звукових хвиль до розуміння змісту питання (Малюнок 5.1).



Малюнок 5.1. Структура знань системи HEARSAY

Знання розподілені по джерелах знань, ● - місце зіставлення даних в умовах, ← місце занесення у діях. Джерело знань – набір модулів типу “умова – дія”:

SEG- перетворить мовні сигнали в дискретну форму, вимірює параметри, утворює сегменти.

POW- на основі сегментів створює гіпотези про склади.

MOW- на основі складів гіпотези про прості слова.

WORD-CTL- керує числом гіпотез, створених MOW.

WORD-SEQ- на основі гіпотези про слова і граматичних правил створює гіпотези про послідовність слів.

WORD-SEQ-CTL- керує числом гіпотез.

PARSE- робить граматичний розбір послідовності слів, якщо усе вірно, створює гіпотези про вираз.

PREDICT- пророкує слова, тобто передують чи випливають на вираз.

VERIFY- оцінює ступінь відповідності між гіпотезою про сегменти і пару зв'язаних слів.

CONCAT- на основі перевірених пар зв'язаних слів створює гіпотезу про вираз.

RPOL- оцінює ступінь довіри іншим гіпотезам на основі інформації в гіпотезах, створених іншими джерелами.

STOP- оцінює необхідність зупинки процесу (кінцева пропозиція чи ні) і вибирає гіпотезу, що вважається найбільш вірною.

SEMANT- інтерпретує зміст для системи пошуку інформації.

Гіпотези оцінюються по шкалі від 0 до 100. Оцінка діє на даному рівні. Малюнок 5.2 демонструє модель дошки оголошень у HEARSAY-II.

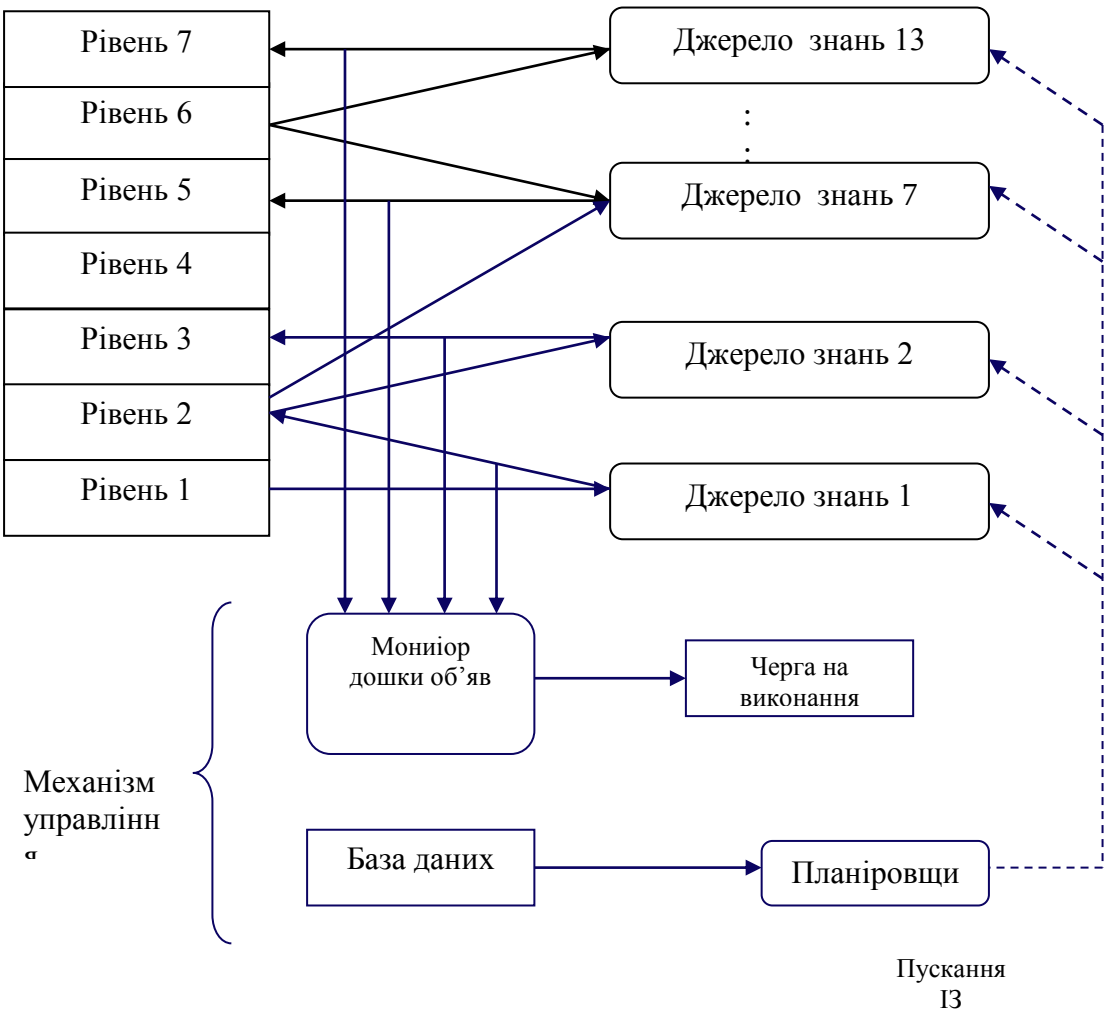


Рисунок 5.2 Модель дошки оголошень у HEARSAY-II

Це розподілена область даних, кожний ІЗ звертається до відповідної області дошки оголошень, вносить гіпотези чи дає їм оцінки в інших областях. Цю систему можна розглядати як розподілену ієрархічну високорівневу продукційну систему. Можна вважати, що усі ІЗ діють асинхронно і паралельно. Планування здійснюється механізмом керування: діє ІЗ з максимальним пріоритетом. Тому при рівнобіжній обробці чи інформації мультиплексорной системи беруть участь у рішенні єдиної задачі незалежно й асинхронно друг від друга.

Контрольні питання

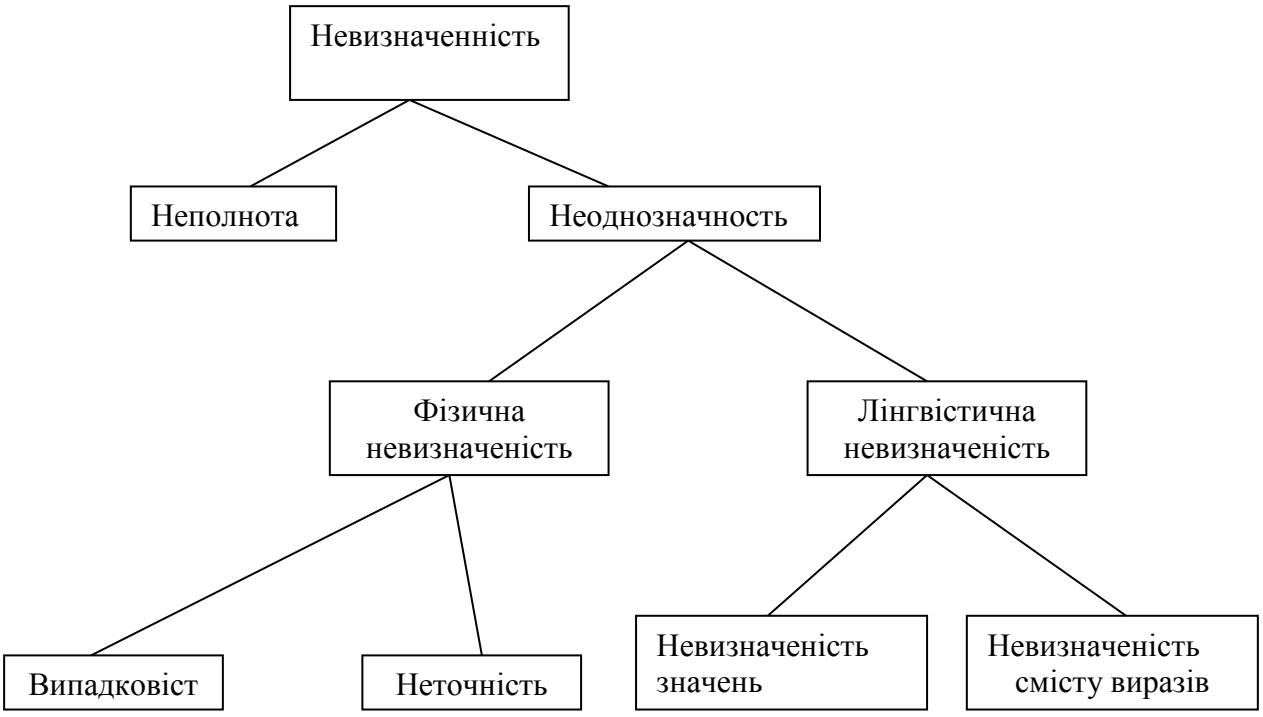
1. Що таке джерело знань у системі з дошкою оголошень?
2. При яких умовах можна вважати виправданим ускладнення системи, використовующей модель з дошкою оголошень.

6. Висновок в умовах чи невизначеності неповних знань

6.2. Види невизначеності

У попередніх прикладах усі знання були визначеними. Твердженнями були чи ІСТИНА, чи НЕПРАВДА. Однак у житті існує тенденція до “нечіткості” у представленні знань. Проте, на підставі неточних даних часто можна робити цілком визначені розумові висновки. Для цього приходиться розглядати комбінацію елементів знань, а також значення їхньої визначеності й у результаті виводити нові знання і давати оцінку їхньої визначеності.

Людина робить необхідні розумові висновки в подібних умовах щодня: ставить медичні діагнози і призначає лікування, керуючись симптомами; з'ясовує причини поганої роботи двигуна по акустичному шумі; вірно розуміє обривки виразів природної мови; довідається друзів по їх голосах і тп. Знаннями, якими оперує людина в зазначених випадках, присуща невизначеність.



Малюнок 6.1. Види невизначеності

Невизначеність може породжуватися неповнотою опису ситуації, ймовірним характером подій, що спостерігаються, неточністю представлення даних, багатозначністю слів природної мови, використанням евристичних правил висновку й ін. Найбільш важливі види невизначеності можна представити у вигляді дерева (Малюнок 6.1) [21].

На першому рівні дерева зображені терміни, що представляють якісну оцінку характеру невизначеності. Невизначеність може бути зв'язана або з неповнотою знань, або з їх неоднозначністю. *Неповнота знань* виникає, коли зібрана не вся інформація, необхідна для побудови висновків. Неоднозначність означає, що істинність тих чи інших висловлень не може бути встановлена з абсолютною вірогідністю. Вона породжується або фізичними причинами (фізична невизначеність), або лінгвістичними (лінгвістична невизначеність). *Фізична невизначеність* може бути зв'язана з випадковістю подій, ситуацій, станів чи об'єкта неточністю представлення даних. *Лінгвістична невизначеність* зв'язана з використанням природної мови для представлення знань, що мають якісний характер, і виникає через множинність значень слів (полісемія) і змісту виразів. Наприклад, "Двигун часто перегрівається" чи "Ваня вже великий". У цих прикладах неоднозначність обумовлена нечіткістю понять "часто" і "великий". Чи, наприклад, "Він зустрів її на галявині з квітами". Як він неї зустрів: з квітами чи без квітів? Такого роду невизначеності присутні в системах, на поводження яких у значній мірі впливають судження людини. При аналізі невизначеності змісту виразів виділяють синтаксичну, семантичну і прагматичну невизначеності [53].

Розглянута схема видів невизначеності деякою мірою умовна. У реальних системах зазначені види невизначеності можуть накладатися один на інший. Наприклад, фізична невизначеність може ускладнюватися лінгвістичною невизначеністю.

Часто зазначені вище види неоднозначності не виділяють в окремі групи, а розглядають у рамках одного терміна "ненадійні знання" [17]. Основним поняттям, використаним при побудові моделей висновку на таких знаннях, є поняття вірогідності. Вірогідність висновків на основі ненадійних знань може бути визначена за допомогою різних підходів. Найбільше часто використовуються: імовірна

байесовська логіка, коефіцієнти впевненості, нечітка логіка, теорія Демпстера-Шефера.

Неповнота знань приводить до необхідності здійснення немонотонних висновків. Для формальної обробки знань, характеризуемих неповнотою, використовуються логіка мовчання Рейтера, немонотонна логіка Мак-Дермотта і Доїла, системи підтримки значень істинності.

6.3. Байесовський метод

При байесовському підході ступінь вірогідності кожного з фактів бази знань оцінюється імовірністю, що приймає значення в діапазоні від 0 до 1. Імовірності вихідних фактів визначають або методом статистичних іспитів, або опитуванням експертів.

Міркування в умовах невизначеності мають місце й у системах спостереження при наявності одночасно декількох конкуруючих гіпотез і їхній постійній переоцінці в міру надходження нових даних. У кінцевому рахунку, виділяється одна гіпотеза, що дозволяє зробити відповідний висновок. По такому принципі працює програма PROSPECTOR, яка застосовується при пошуку рудних родовищ.

Системи розпізнавання мови також використовують цей метод. Розпізнавання мови - складна задача, тому в ній повинні бути присутніми конкуруючі гіпотези, наприклад, про те, яке конкретне слово вживається в пропозиції. На основі наявних гіпотез за допомогою різних джерел неповної інформації робиться опосередкований висновок.

Якщо існує тільки одна застосовна евристика, то проблем немає. Але що робити, якщо два евристичних методи орієнтують програму в двох різних напрямках? А коли дві евристики вказують той самий напрямок пошуку, чи повинне це викликати більшу довіру, чим, якби була тільки одна з них?

Існують чотири важливі проблеми, які необхідно обговорити стосовно до поняття невизначеності в автоматичних системах висновку:

1. Як кількісно виразити ступінь визначеності при встановленні істинності (чи хибності) деякої частини даних?

2. Як виразити ступінь підтримки висновку конкретною посилкою?

3. Як використовувати спільно двох (чи більш) посилок, що незалежно впливають на висновок?

4. Як бути в ситуації, коли потрібно обговорити ланцюжок висновку для підтвердження висновку в умовах невизначеності?

В усіх міркуваннях в умовах невизначеності використовуються чотири основні правила [30].

Умовна імовірность події А при даному В – це імовірность того, що подія А наступить за умови, що наступила подія В. Наприклад, імовірность того, що пацієнт дійсно страждає захворюванням А, якщо в нього виявлений тільки симптом У:

$$p(A|B)=p(A \text{ і } B)/p(У) \text{ чи } p(B|A)=p(A \text{ і } B)/p(A).$$

Це - основна формула умовної імовірності.

Якщо для кожної формули обчислити величину p(A і B) і прирівняти результати, то вийде:

$$p(A)*p(B|A)=p(У)*p(A|B)$$

Ця формула відома як *правило Байеса*.

Наступне правило І/ЧИ:

$$p(A \text{ чи } B)-p(A)+ p(У)-p(A \text{ і } B).$$

Ще одне правило називається *правилом композиції*. У ньому затверджується, що імовірність події А є середнє зважене двох інших імовірностей:

$$p(A)-p(A|B)*p(У)+ p(A|\text{ні } B)*p(\text{ні } B).$$

Розглянемо першу ситуацію, у якій використовується правило типу

якщо (А), то (У)

Припустимо, що ніякі інші правила не мають відносини до даної ситуації. Розберемося, коли можна зробити висновок, що В істинно.

Де виникає невизначеність? У системах висновку вона буває двох видів. Так, невизначеність виникає при спробі кількісно оцінити, наскільки ми упевнені, що попередня умова щира. Наприклад, якщо ступінь упевненості того, що А істинно, складає тільки 90%, то яке значення тоді прийме В?

Інший варіант - невизначеність у самій імплікації. Наприклад, можна сказати, що в більшості випадків, але не завжди, якщо є А, тобто також і В. Повинне бути числови вираз цієї події (скажемо, на 95% упевнені, що, маючи А, маємо й У).

Як можна всі ці відносини виразити в термінах імовірності? Якби була абсолютна гарантія, що попередня подія А щира, то можна записати:

$$p(A)=1.$$

Коли повної визначеності немає, встановлене значення імовірності відбиває цю інформацію в такий спосіб:

$$p(A) = 9.$$

При невизначеності другого типу твердження з імовірністю в 95%, що буде В, якщо є А, записується у формі

$$p(B|A)\text{-}95.$$

Тут використана умовна імовірність. Це формулювання досить ясне саме по собі, але воно не дає ніякої інформації про те, чи може бути В, якщо немає А. У ряді випадків така можливість є, і тоді потрібно одержати значення В при відсутності А:

Розглянемо тепер типову проблему, що включає просту імплікацію. Імплікація являє собою вираження типу:

якщо (А), то (У)

Проблему можна виразити в такий спосіб: ми на 90% упевнені в тім, що А істинно. Ми віримо в це правило на 95%. Яка імовірність, що В істинно?

Відомо наступне:

p(A)-.9.;

p(B|A)-.95.;

p(У) — шуканий результат.

Як обчислити p(У)? Можна застосувати правило композиції і з його допомогою одержати p(У) на основі відомих імовірностей:

$$p(У)=p(У\ |A)\ *p(A)+\ p(У\ |ні\ A)\ *p(ні\ A).$$

Три величини відомо. Зверніть увагу на те, що:

$$p(\text{ні A}) = (1—p(A)) = 0.1.$$

Тому після перетворення уже відомих величин виходить рівність:

Р(У)=.95*.9+ р(В|ні А)*.1-.855+ р(У |не А)*.1.

До цього місця обчислення проводилися без додаткової інформації. Тому що не відомий фактор укладений між 0 і 1, то відповіддю на основне питання буде твердження, що:

$$0.855<-p(У)<-0.955.$$

Якщо при постановці проблеми малося на увазі (але не повідомлялося), що В ніколи не є присутнім без А, то

$$p(У \mid \text{ні А}) = 0,$$

і можна точно сказати, що

$$p(У) -0 .85.$$

В іншій ситуації для одержання висновку можуть бути присутні дві посилки:

якщо (А і В), то (С).

Умовна імовірність знову дає всі необхідні зведення для міркування в цій ситуації. На жаль, при постановці задачі вказується недостатня кількість даних, що не дозволяє прийти до точної відповіді.

От типова проблема. Наприклад, упевнені на 95%, що А і В спричиняють С. При цьому з імовірністю в 80% відомо, що А істинно, і з імовірністю в 70% - що В істинно. Яка імовірність С ?

Відомо наступне:

р(А)-.8;

Р(У)-.7;

р(С|А й У)-.95;

р(У) - шуканий результат.

Якщо слідувати формулюванням попередньої задачі, то при використанні правила композиції вийдуть такі вираження для С:

p(C) = p(C|А й У)*p(C| ні(А і В))*p(ні(А і В)),

p(C) = .95*p(А і В)+ p(C |ні(А і В))*(1—p(А і В)).

Тепер маємо дві невідомі величини замість однієї.

При таких міркуваннях вирішальне значення набуває співвідношення між p(А) і p(У). На жаль, як стане ясно з подальшого, зазначене співвідношення не можна обчислити. Неможливо одержати p(А і В) на основі імовірностей компонент, хоча експерти – не математики часто формулюють свої правила так, начебто подібний зв'язок існує.

Приведений вище приклад показує, що прості імовірності компонентів дають мало інформації про можливість спільного настання події. Однак вони вказують інтервал, у який ця імовірність обов'язково потрапить.

Якщо на цій основі потрібно зробити висновок, причому не можна одержати додаткову інформацію, тоді, імовірно, найкращими виявляться наступні припущення:

p(C І ні (А і В))=0,

і p(А і В) має середнє значення в припустимому інтервалі.

Однак помітьте, що почали міркувати евристично, відмовившись від математичної точності.

Існує ще один тип імплікації, що зустрічається звичайно в системах висновку

якщо (А чи В), то (С)

Якщо параметри виразу не визначені, то що можна сказати про С?

Типову задачу можна сформулювати в такий спосіб:

Якщо істинно тільки А, у 70% випадків С також буде щирим.

Якщо істинно тільки В, у 80% випадків С також буде щирим.

Якщо істинно й А, і В, у 95% випадків С також буде щирим.

Крім того, у даній конкретній ситуації на 80% упевнені, що А істинно, і на 80% упевнені, що В істинно.

Яка імовірність істинності С?

Сформулюємо в термінах теорії імовірності:

p(C | A і (ні у)) =.7;

p(C | (ні A) і B) = .8;

p(C І A й У))=.95;

p(A) = .8;

p(У) = .8.

Користуючись рішеннями попередніх задач, запишемо вираження для p(C), перелічивши всі умови, коли може відбутися подія С (звертаємося до правила композиції):

p(C) = p(C | A і B)*p(A і B)

+ p(C | A і (ні B))*p(A і (ні B))
+ p(C | (ні A) і B) * p((ні A) і B)
+ p(C | (ні A) і (не B)) *p((ні A) і (ні B)).

Всі інші умови включають інтервали, що обчислюються за допомогою лежачих у їхній основі конкретних імовірностей, як робилося в попередній задачі.

Якщо на основі цих формальних викладень усе-таки необхідно зробити висновок, то залишається лише сподіватися, що наступна рівність справедлива:

p(C | (ні A) і (ні B))=0.

Є ще чотири не відомі імовірності, для яких можна обчислити інтервали. Найкращою оцінкою такої величини розумно вважати середину припустимого для неї інтервалу.

Таким чином, що міркування строге на основі імовірностей стає усе більш важким і незручним.

З цієї причини багато експертних систем не використовують міркування на основі умовної імовірності і виконують лише грубі оцінки умов чи імовірності виробляють деякі спеціальні схеми, що відображають те, що міг би зробити експерт.

Можна створити багато різних схем наближеного міркування. Нижче розглянемо одну з найважливіших схем - механізм міркування медичної системи висновку MYCYN, перетворену в більш загальну модель EMYCYN, що і стала закінченою формою цієї програми.

Розглянемо простий тип імплікації:

якщо (E), то (C)

Ефективний спосіб рішення - привласнити коефіцієнт визначеності як посилці, так і всієї імплікації. Тоді можна спільно використовувати ці дві величини для обчислення коефіцієнтів визначеності усього висновку.

Що таке коефіцієнт визначеності? Його часто застосовують замість поняття імовірності. Якщо позначити коефіцієнт визначеності як ct, а імовірність як p, то в найпростішому випадку одержимо:

коефіцієнт визначеності посилки - ct (E) ~ p (E),

коефіцієнт визначеності імплікації - ct (C) ~ p(C/E).

Таким чином, коефіцієнт визначеності події приблизно еквівалентнийі імовірності того, що посилка (як затверджується) є щирої. Коефіцієнт визначеності імплікації подібний з умовною імовірністю висновку, отриманого при істинності посилки.

Звичайне правило комбінування, що дозволяє обчислити коефіцієнт визначеності висновку у випадку, коли відомий коефіцієнт визначеності посилки, що лежить у його основі, і зв'язку в імплікації, записується так:

ct (висновок) = ct (посилка) *ct (імплікація).

Саме такий механізм використаний у EMYCYN. Наприклад, віримо в істинність посилки з імовірністю .8. Віримо й у те, що лежача в основі імплікації схема виконується в більшості випадків, але не завжди. Тому приписуємо їй коефіцієнт визначеності .9. Тоді коефіцієнт визначеності висновку в такій ситуації

ct (висновок) = .8 * .9 = .72.

Насамперед потрібно зуміти оцінити коефіцієнти визначеності посилок. Будемо називати посилкою все логічне вираження в правилі між "якщо" і "те". За винятком випадків простої імплікації, це вираження складається з атомарних посилок, кожна з який має свій коефіцієнт визначеності. Вони можуть бути зв'язані між собою логічними операціями, наприклад:

якщо(e1 чи (e2 і e3)), то (c)

чи

якщо (e1 і e2 і ((ні e3) чи e4)), то (c)

Очевидно, потрібно деякий спосіб оцінки коефіцієнтів визначеності цих складних форм у поняттях їхніх окремих компонентів.

Підхід, використаний у EMYCYN, полягає в тім, щоб відкинути всі складні вираження і вважати всі правила простими. Таке обмеження, проте зберігає структури правил, що є досить інформативними для більшості цілей. Є кілька тривіальних процедур для зведення коефіцієнтів визначеності простих логічних комбінацій в одне число.

Найпростішою логічною комбінацією є кон’юнкція (I) між двома елементарними свідченнями:

якщо (e1 і e2), то (c)

Відповідно до оцінки, зробленої в EMYCYN, коефіцієнт визначеності посилки дорівнює коефіцієнту визначеності найменш надійної з посилок, тобто

ct(el і e2) = min (ct(el),ct(e2))

Іншою простою формою є правило, у якому використовується диз'юнкція (ЧИ), що зв'язує дві частини свідчень:

якщо (e1 чи e2), то (c)

Загальне правило комбінування, по якому обчислюється коефіцієнт визначеності посилки, полягає в тім, що коефіцієнт визначеності диз'юнкції дорівнює коефіцієнту визначеності її найсильнішої частини, тобто

C(e1 чи e2) = мах (c1(e1),c1(e2))

Хоча правила іноді і записуються за допомогою диз'юнкції, якщо є вибір, то прийнято розбивати диз'юнкцію на дві частини, наприклад:

якщо (e1),то (c)

якщо (e2), то (c)

Використання двох правил замість диз'юнкції дозволяє більш чітко побачити ситуацію, але якщо необхідно дотримуватись цього розуміння, то потрібний механізм, що визначає коефіцієнт визначеності висновку за підтримкою двох правил.

Розглянемо ситуацію, коли використовуються два правила й обоє вони підтримують тий самий висновок, наприклад:

Правило 1:
якщо (e1), то (c) ct (висновок) = .9

Правило 2:
якщо (e2) то (c) ct (висновок) = .8

Допустимо, обидві посилки вірні, і обчислили імовірність висновку для кожного правила по окремості.

Якщо використовувалося правило 1, то коефіцієнт визначеності у висновку виявиться рівним 0.9. Ясно, що, маючи ще і правило 2, одержимо більший коефіцієнт визначеності, але яка буде його величина?

Припустимо, що змінна ctotal представляє загальний коефіцієнт визначеності висновку, отриманий використанням усіх підтримуючих його правил. Можна запропонувати багато різних комбінацій процедур. У EMYCYN, наприклад, діє простий і ефективний механізм:

ctotal = коефіцієнт визначеності з правила 1

+ коефіцієнт визначеності з правила 2

— (коефіцієнт визначеності з правила 1)

* (коефіцієнт визначеності з правила 2)

Підставивши числа, задані в прикладі, одержимо

ctolal = .9 + .8 — (.9) * (.8) = .98.

Розглянутий принцип можна поширити на випадок n правил, що підтримують один висновок, де для кожного правила існує своя імовірність. Наприклад, у нас є три правила з наступними коефіцієнтами визначеності:

Правило 1:

якщо (el), то (с) ct (висновок) = ctl

Правило 2:

якщо (е2), то (с) ct (висновок) = ct2

Правило 3:

якщо (ез), то (с) ct (висновок) = ct3

Сукупний коефіцієнт визначеності висновку з обліком усієї можливої підтримки може бути обчислений так:

ctotal =ctl+ct2+cl3— ctl * ct2 — ct2 * ct3 — ctl * ct3 + ctl * ct2 * ct3

Таким чином, коефіцієнт визначеності- це артефакт приблизних міркувань. Немає іншого доказу правомірності такого способу комбінування, крім того, що він простий, відповідає здоровому глузду і випливає правильному загальному поводженню, якщо не відноситися до нього зайво довірливо.

Механізм доповнення- це інший спосіб обчислення коефіцієнта визначеності висновку, підтримуваного декількома правилами імплікації. Він використовується у випадку, коли зведення про дозволені до застосування правила надходять послідовно, а не одночасно. Наприклад, якщо система задає користувачу питання, то застосування нових правил буде відбуватися по черзі.

Розглянемо приклад. Допустимо, відомо, що висновок підтримується двома правилами з наступними коефіцієнтами визначеності:

Правило 1:

якщо (е1), то (с) ct (висновок) = ctl.

Правило 2:

якщо (e2), то (c) ct (висновок) = ct2.

При застосуванні двох правил сукупний коефіцієнт визначеності

ctotal -= ctl + ct2 — ctl *ct2.

Тепер припустимо, що з'явилося третє правило, що підтримує той же висновок:

Правило 3:

якщо (e3), то (c) ct(висновок) = ct3

Якщо усе, що отримано з попереднього дослідження, входить у змінну ctotal, і якщо вважається, що ct3 може ввійти в міркування на загальних підставах, то можна використовувати стратегію доповнення для формування зміненої оцінки коефіцієнта визначеності висновку;

cnewtotal = ct3 + ctotal — ct3 * ctotal.

У будь-якому випадку при використанні механізму доповнення черга надходження правил, що підтримують висновок, не має значення.

Можна поєднувати коефіцієнти визначеності з підтримуючих імплікацій послідовно в міру їхнього надходження чи зберігати інформацію, а потім використовувати її усю відразу - результат від цього не міняється.

Практично мережа міркування міняється, як тільки надходять нові зведення. Тому зберігати потрібно лише сукупний коефіцієнт визначеності для кожного висновку, що забезпечує найбільш ощадливий спосіб підтримки інформаційного забезпечення ЭС.

6.4. Біполярні схеми для коефіцієнтів визначеності

Прототипом систем, заснованих на наближених міркуваннях, є MYCYN і її прямий нащадок EMYCYN. У EMYCYN у будь-якому випадку, коли повинна бути чисельно виражена визначеність, використовується інтервал від -1 до + 1, так що це не може бути імовірністю. Границі інтервалу позначають наступне: +1- система в чомусь цілком визначена, 0- у системи немає знань про обговорювану величину, -1-

висловлена гіпотетична посилка чи висновок абсолютно невірно. Проміжні величини відбивають ступінь довіри чи недовіри до зазначених ситуацій.

Всі описані процедури міркуванні застосовні і для коефіцієнтів визначеності, що задаються в цих більш широких границях.

Повна реалізація ідеї біполярних коефіцієнтів визначеності вимагає зробити два узагальнення.

Перше, якщо в правилі є заперечення, наприклад:

якщо (e1 і (ні e2)), то (c)

те потрібно вважати (ні e2) новим атомарним твердженням, наприклад e3, Для обчислення ж коефіцієнта визначеності (ні e2) досить просто поміняти знак:

c1(ні e) = - ct(e)

Інше– правило одержання коефіцієнтів визначенностей в умовах підтримки двома правилами того самого висновку:

якщо обидва коефіцієнти визначеності позитивні, то:

ctotal = ctl + ct2 — ctl * ct2,

якщо обидва коефіцієнти визначеності негативні, то:

ctotol = ctl + ct2 + ctl * ct2,

Коли негативний один коефіцієнт, то:

ctotal = (ctl+ct2)/(l —min(abs(ctl),abs(ct2)))

У тому випадку, коли одна визначеність дорівнює +1, а інша-1,

Ctotal = 0.

Таблиця 6.1 містить усі можливі способи комбінування двох коефіцієнтів визначеності відповідно до зазначеного вище правилами. Значення і знаки на цьому рисунку нку відповідають здоровому глузду.

Таблиця 6.1 Результат композиції коефіцієнтів визначеності

	-1	**-0.8**	**-0.6**	**-0.4**	**-0.2**	**0**	**0.2**	**0.4**	**0.6**	**0.8**	**1**
1.0	0	1	1	1	1	1	1	1	1	1	1
0.8	-1	0	0.5	0.67	0.75	0.8	0.84	0.88	0.92	0.96	1

0.6	-1	-0.5	0	0.33	0.5	0.6	0.68	0.76	0.84	0.92	1
0.4	-1	-0.67	0.33	0	0.25	0.4	0.52	0.64	0.76	0.88	1
0.2	-1	-0.75	-0.5	-0.25	0	0.2	0.36	0.52	0.68	0.84	1
0	-1	-0.8	-0.6	-0.4	-0.2	0	0.2	0.4	0.6	0.8	1
-0.2	-1	-0.84	0.68	-0.52	-0.36	-0.2	0	0.25	0.5	0.75	1
-0.4	-1	-0.88	-0.76	-0.64	-0.52	-0.4	-0.25	0	0.33	0.67	1
-0.6	-1	-0.92	-0.84	-0.76	-0.64	-0.6	-0.5	-0.33	0	0.5	1
-0.8	-1	-0.96	-0.92	-0.88	-0.84	-0.8	-0.75	-0.67	-0.5	0	1
-1.0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0

Значення коефіцієнтів визначеності для правила 1 відкладені по горизонтальній осі, а для правила 2- по вертикальній. Числа відбивають результати конкретних комбінацій. Тут відбувається наступне: коли два правила з невеликими коефіцієнтами визначеності підтримують один висновок, коефіцієнт визначеності висновку зростає. Якщо ж знаки не збігаються, то результат визначається найсильнішим, але вплив його трохи послабляється. Застосування біполярних коефіцієнтів визначеності може привести до нереальних результатів, якщо правила сформульовані неточно. Працюючи з одним правилом висновку завжди використовується співвідношення:

c1 (висновок) ~ct (посилка) *ct (імплікація).

Усі правила попадають в одну з цих двох дуже важливих категорій.

Правила першої категорії будемо називати оборотними. Правило вважається оборотним, якщо при додаванні заперечення не і до умови і до висновку воно не втрачає зміст. Однією з характеристик такого правила є його застосовність до будь-якому вероятностного значення, що може бути зв'язане з посилкою.

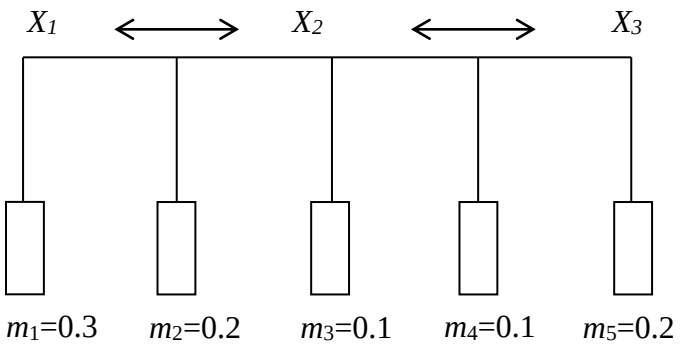
Правила другої категорії вважаються необоротними. Ці правила "працюють" тільки при позитивних значеннях посилки. Якщо ж її значення негативне, правило застосовувати не можна.

6.5. Теорія свідчень Демпстера-Шефера

Підхід, прийнятий у теорії Демпстера-Шефера (ТДШ) [21] відрізняється від байесовского підходу і методу коефіцієнтів упевненості тим, що, по-перше, тут використовується не крапкова оцінка впевненості (коефіцієнт упевненості), а интервальная оцінка. Така оцінка характеризується нижньою і верхньою границею, що більш надійно. По-друге, ТДШ дозволяє виключити взаємозв'язок між невизначеністю (неповнотою знань) і недовірою, що властива байесовскому підходу.

У рамках ТДШ безлічі висловлень *A* приписується діапазон значень *[Bl(A),P1(A)],* у якому знаходяться ступені довіри (правдоподібності) кожного з висловлень. Тут *У1(A)* - ступінь довіри до безлічі висловлень, що змінює свої значення від 0 (немає свідчень на користь *A*) до 1 (безліч висловлень *A* істинно); *P1(A)* - ступінь правдоподібності безлічі висловлень *A,* обумовлена за допомогою формули: Pl(A) = 1-Bl(not A).

Припустимо, що існують дві конкуруючі гіпотези h_1 і h_2. При відсутності інформації, що підтримує ці гіпотези, міра довіри і правдоподібності кожної з них належить відрізку [0; 1]. В міру нагромадження ці інтервали будуть зменшуватися, а довіра гіпотезам – збільшуватися. У теорії Демпстера-Шефера невизначеність знань представляється за допомогою деякої безлічі *X*. Елементи цієї безлічі відповідають можливим фактам чи висновкам. Невизначеність полягає в тому, що заздалегідь невідомо, яке з можливих значень прийме факт чи висновок $x \in X$. Для характеристики ступені визначеності в ТДШ уводиться деяка одинична міра впевненості (її називають також одиничною масою впевненості), що розподіляється між елементами *X*. При цьому, якщо вся маса (ступінь) упевненості приходиться на один елемент $x \in X$, то ніякої невизначеності немає. Невизначеність виникає, коли маса впевненості розподіляється між декількома елементами $x \in X$. Розподіл мас упевненості (Малюнок 6.2) між елементами безлічі *X*, представлено у виді крапок [21].Тут $X=\{x_1, x_2, x_3\}$.



Малюнок 6.2. Розподіл мас упевненості

З кожним елементом безлічі X жорстко зв'язана відповідна маса впевненості. Так, x_1 відповідає m_1= 0,3 x_2-m_2= 0,1 x_3-m_3= 0,2. Маються також вільні маси впевненості m_4= 0,2 m_5= 0,2, що відносяться відразу до декількох елементів. Маса m_4 вільно переміщається між елементами x_1 і x_2, а маса m_5 - між елементами x_2 і x_3, тобто m_4 закріплена за підмножиною $\{x_1, x_2\}$, а m_5 - за підмножиною $\{x_2, x_3, \}$. Маси виражають ступінь впевненості в можливих значеннях чи фактах висновків. Так, ступінь впевненості в значенні x_1 може змінюватися від 0,3 до 0,5. Таким чином, ступінь незнання відповідає масі, місце розташування якої не визначено.

У загальному випадку розподіл мас упевненості задається функцією *m(A)*, що володіє наступними властивостями:

$$m(\varnothing)=0,$$
$$\sum m(A)=1,$$

Тут A - безліч, утворена з підмножин X, яким призначені відповідні маси (ступеня) упевненості; *m(A)*- функція, що задає відображення A на інтервал [0, 1]. Для приклада (Малюнок 6.2) маємо :

$$A = \{\varnothing, \{x_1\}, \{x_2\}, \{x_3\}, \{x_1,x_2\}, \{x_2,x_3\}, \{x_1,x_3\}\},$$

а розподіл мас упевненості задається функцією *m(A)*, характеризуемой безліччю значень:

$$m(A) = \{0; 0,3; 0,1; 0,2; 0,2; 0,2; 0\}.$$

Звернемо увагу, що A складається з підмножин. Позначимо кожну таку підмножину через A_i. Ступінь довіри до висловлень, що відповідають підмножині A_i, може бути обчислена по формулі

$$Bl(\{A_i\}) = \sum_{A_j \subseteq A_i} m(A)$$

Тут підсумовування виконується по всіх інших підмножинах A_j вхідним A_1 . Наприклад:

$$Bl(\{x_1,x_2\}) = m(A_1)+m(A_2)+m(A_3)=m(\{x_1\})+m(\{x_2\})+m(\{x_1, x_2\})=$$
$$0.3+0.1+0.2=0.6$$

Результати обчислень ступенів правдоподібності дана нижче (Таблиця 6.2).

Таблиця 6.2 Значення $Bl(A_i)$ *і* $Pl(A_i)$

A_i	$\emptyset$	$\{x_1\}$	$\{x_2\}$	$\{x_3\}$	$\{x_1,x_2\}$	$\{x_2,x_3\}$	$\{x_1,x_3\}$	X
$Bl(A_i)$	0	0.3	0.1	0.2	0.6	0.5	0.5	1
$Pl(A_i)$	0	0.5	0.5	0.4	0.8	0.7	0.9	1

Ступінь правдоподібності підмножини A_i визначається по формулі:

$$Pl(A_i) = 1 - Bl(notA_i) = 1 - \sum_{A_j \in A_i} m(A_j)$$

Величини $Bl(A_i)$ *і* $Pl(A_i)$ мають просту інтерпретацію: $Bl(A_i)$ представляє загальну масу впевненості, що залишається, якщо з X видалити всі елементи, не асоціюемі з A_i. $Pl(A_i)$ представляє максимальну масу впевненості, яку можна одержати, якщо зрушити вільні маси до елементів безлічі A_i. Причому $Bl(A_i) \le Pl(A_i)$. Іншими словами, $Bl(A_i)$ представляє нижню границю довіри до A_i, а $Pl(A_i)$ - верхнього.

Найважливішим елементом ТДШ є правило комбінації свідчень:

$$m(A_k) = \frac{\sum_{A_{1i} \cap A_{2j}} m_1(A_{1i}) \cdot m_2(A_{2j})}{1 - \sum_{A_{1i} \cap A_{2j} = \varphi} m_1(A_{1i}) \cdot m_2(A_{2j})}$$

Сума в чисельнику правила поширюється на безліч $A_k = A_{1i} \cap A_{2j}$. Правило є евристичним і дозволяє здійснювати розподіл ступенів довіри в ході висновку. Наприклад, мірою довіри $m_n(Z)$ гіпотезі Z – для n=3 джерел свідчень вважається сума добутків гіпотетичних мір довіри $m_1(X)$ і $m_2(Y)$, спільне входження яких підтримує Z, тобто $X \cap Y = Z$. Знаменник у правилі Демстера допускає порожнє перетинання $X\ Y$, а сума мір довіри повинна бути нормалізована.

Розглянемо застосування правила Демпстера для задачі медичної діагностики, описане в [36].

Припустимо, що розглядається область Q, що містить чотири гіпотези:

1. пацієнт був без свідомості (C);
2. у нього був грип (F);
3. мігрень (H);
4. менінгіт (M).

Необхідно зв'язати міри довіри з безліччю гіпотез у рамках Q. Наприклад, лихоманка свідчить на користь {C,F,M}. Тому що елементи Q трактуються як взаємовиключні гіпотези, підтвердження однієї з них може впливати на вірогідність інших.

Нехай є свідчення, що в пацієнта лихоманка. Воно підтримує {C,F,M} з імовірністю 0,6. Назвемо це першою мірою довіри m_1. Якщо це усього лише гіпотеза, то $m_1\{C,F,M\}=0,6$, де $m_1\{Q\}=0,4$ залишок (1-0.6) частини розподілу, що залишилася, вірогідності, тобто всі інші можливі міри довіри Q, а не вірогідність доповнення {C,F,M}.

Потім був отриман факт про новий прояв хвороби- у пацієнта блювота, що свідчить про {C,F,H} зі ступенем довіри 0,7. Нехай це буде міра довіри свідчення m_2. Тоді маємо $m_2\{C,F,H\}=0,7$, де $m_2\{Q\}=0,3$.

Одержуємо в такий спосіб безліч X – набір підмножин Q на який m_1 приймає ненульові значення, і Y - набір підмножин Q на який m_2 приймає ненульові значення.

Застосуємо правило Демпстера [36] для визначення об'єднаної міри довіри m_3: перемножимо X і Y. Знаменник дорівнює 1, тому що поки не існує порожніх безлічей $X \cap Y$. Результат обчислень Таблиця 6.3.

Таблиця 6.3.Застосування правила Демстера для об'єднання свідчень m_1 і m_2

$\mathbf{m_1}$	$\mathbf{m_2}$	$\mathbf{m_3}$
$m_1\{\mathbf{C},\mathbf{F},M\}=0,6$	$m_2\{\mathbf{C,F},H\}=0,7$	$m_2\{\mathbf{C,F}\}=0,42$
$m_1\{Q\}=0,4$	$m_2\{C,F,H\}=0,7$	$m_2\{C,F,H\}=0,28$
$m_1\{C,F,M\}=0,6$	$m_2\{Q\}=0,3$	$m_2\{C,F,M\}=0,18$
$m_1\{\mathbf{Q}\}=0,4$	$m_2\{\mathbf{Q}\}=0,3$	$m_3\{\mathbf{Q}\}=0,12$

Зверніть увагу на міркування й угруповання гіпотез. Чотири безлічі стовпця m_3 являють собою всі можливі перетинання X і Y. Цих даних явно недостатньо для установки діагнозу, що і відбивають отримані числа.

Додамо дані лабораторного аналізу, що свідчить на користь менінгіту $m_4\{M\}=0,8$ і $m_4\{Q\}=0,2$.

Застосуємо ще раз правило Демпстера [36] для визначення об'єднаної міри довіри m_5. Результат обчислень Таблиця 6.4.

Таблиця $m_5\{M\}$ виходить у декількох випадках, то загальна імовірність $m_5\{M\}=(0,144+0,096)=0,240$.

У результаті перетинання декількох пар безлічей виходить порожня безліч {}, значить знаменник у рівняння Демпстера потрібно вважати як

(1-(0,336+0,224))=0,44.

Таблиця 6.4 Застосування правила Демстера для об'єднання свідчень m_3 і m_4

m_3	**m_4**	**m_5**
m_2{C,F}=0,42	m_4{M}=0,8	M_5{}=0,336
m_3{Q}=0,12	m_4{M}=0,8	M_5{M}=0,096
m_2{C,F}=0,42	m_4{Q}=0,2	M_5{C,F}=0,084
m_3{Q}=0,12	m_4{Q}=0,2	M_5{Q}=0,024
m_2{C,F,H}=0,28	m_4{M}=0,8	M_5{}=0,224
m_2{C,F,M}=0,18	m_4{M}=0,8	M_5{M}=0,144
m_2{C,F,H}=0,28	m_4{Q}=0,2	M_5{C,F,H}=0,056
m_2{C,F,M}=0,18	m_4{Q}=0,2	M_5{C,F,M}=0,036

Остаточні значення міри довіри мають вид:

m_5{M}=0,240/0.44=0.545 m_5{C,F}=0.084/0.44=0.191

m_5{C,F,H}=0.056/0.44=0.127 m_5{C,F,M}=0,82

m_5{}=0,336+0.224=0.56

m_5{Q}=0,024/0.44=0.055

Висока вірогідність порожньої безлічі m_5{}=0.56 означає існування конфлікту свідчень на безлічі мір довіри m_j тому що в прикладі дані некоректні з погляду медицини дані.

При існуванні великих безлічей гіпотез обчислення мір довіри може виявитися громіздким, але все-таки значно менше ніж при використанні теореми Байеса (розділ 6.2).

Правило Демстера- приклад міркувань суб'єктивних імовірностей, на відміну від об'єктивних імовірностей Байеса.

6.6. Нечіткі безлічі і нечітка логіка

Для формалізації нечітких знань, характеризуемих лінгвістичною невизначеністю, застосовується теорія нечітких чи розпливчастих безлічей. Основи теорії нечітких безлічей були створені в 1965 році Л.А. Заде (США).

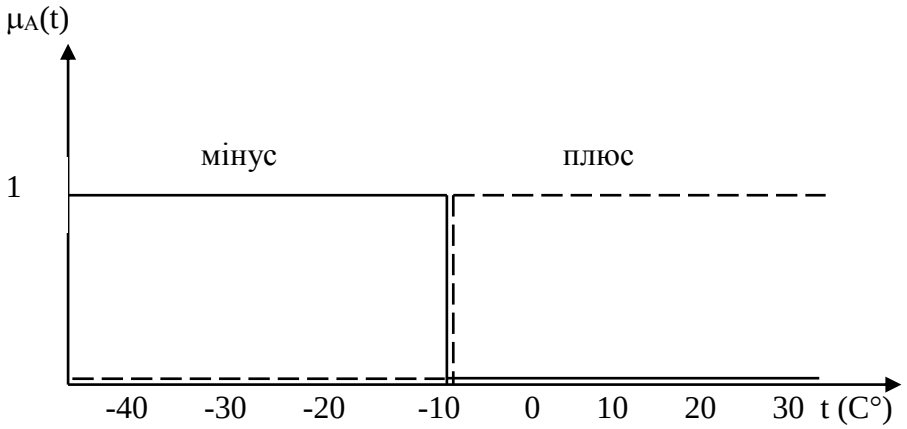
Нехай мається безліч X і його власна безліч A, тобто $A \subset X$. Тоді підмножину A можна представити у виді сукупності упорядкованих пар:

$$A = \{(x, \mu_A(x))\}, \forall x \in X$$

де $\mu_A(x)$ функція приналежності. Причому:

$$\mu_A(x) = \begin{cases} 1, если \cdot x \in A \\ 0, если \cdot x \notin A \end{cases}$$

Розглянемо як приклад [21] безліч, що представляє можливі значення температури повітря. Тоді уведення функції приналежності, умовно зображеної на малюнку 4.3 суцільною лінією, дозволить виділити підмножину негативних температур, а зображеною пунктирною лінією- підмножину позитивних температур. При цьому нульова температура відноситься до другої підмножини.



Малюнок 6.3. Функція приналежності для чіткої безлічі

Приведене визначення підмножини А і приклад функції приналежності, що приймає усього два можливих значення 0 і 1, відповідає чіткому (звичайному) підмножиню. Визначення нечіткої підмножини виходить як узагальнення цього визначення. Нечіткою підмножиною *A* безлічі *X* будемо називати сукупність

$$A = \{(x, \mu_A(x)\}, \forall x \in X$$

упорядкованих пар:

де функція приналежності $\mu_A(x)$ кожному елементу *x* ставить у відповідність дійсне число з інтервалу [0, 1], що вказує ступінь приналежності елемента *x* підмножині *A*.

Математична структура, обумовлена вираженням

$$A = \{(x_1 |0{,}8) , (x_2 |0{,}3), (x_3 |0), (x_4 |0{,}5)\},$$

де x_1, x_2, x_3, x_4 - елементи універсальної безлічі *X*, являють собою приклад нечіткої підмножини. Тут ступеня приналежності елементів x_1, x_2, x_3, x_4 підмножині *A* задані числами після вертикальної риси. Найвищий ступінь приналежності має елемент x_1. Елемент x_3 підмножині *A* не належить. Елементи x_2 і x_4 належать *A* у меншій ступені, чим x_1 . Таким чином, використовуючи поняття нечіткої підмножини, можна представляти об'єкти (сутності) предметної області, характеризуємі розмитими границями описів.

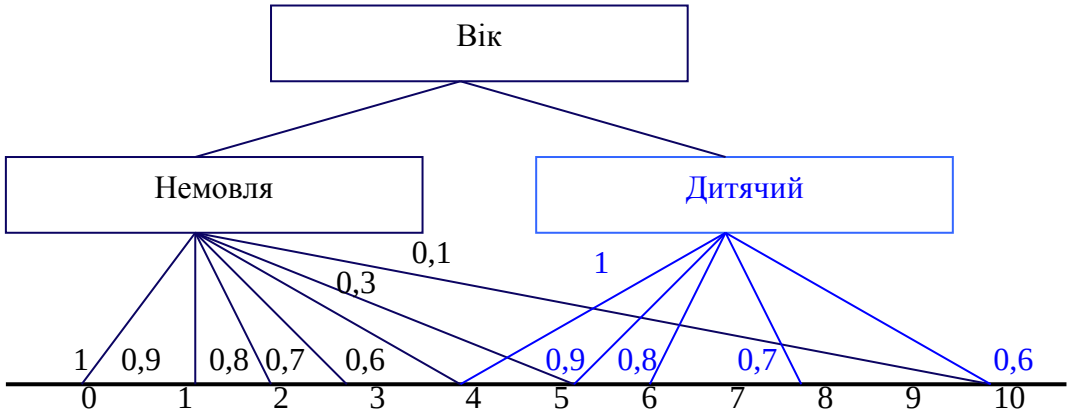
Нечітка безліч A називається *нормальной* , якщо

$$\max_{x \in X} \mu_A(X) = 1$$

Якщо ця властивість не виконується, то A називається *субнормальной*. Непорожню підмножину A завжди можна нормалізувати розподілом на максимальне значення функції приналежності. Носієм нечіткої підмножини A називається підмножина елементів X, для яких $\mu_A(x) > 0$. Змінна x називається *базовою.*

При практичному застосуванні нечітких безлічей важливим є поняття нечіткої і лінгвістичної змінної [65]. Лінгвістична змінна (ЛП) – це змінна, значення якої визначається набором вербальних характеристик деякої властивості. Наприклад «з*ріст*» визначається через набір {*карликовий, низький, середній, високий, дуже високий*}.

Приведемо приклад з [65] інтерпретації значень ЛП «*вік*», що може бути визначений через набір {*дитячий, дитячий, юний, молодий, зрілий, старий*}. Для ЛП «*вік*» базова шкала від 0 до 120 років, що позначає кількість прожитих років, а функція приналежності визначає, наскільки ми упевнені в тім, що дану кількість років можна віднести до даної категорії віку. Малюнок 6.4 показує, як ті самі значення базової шкали можуть брати участь у визначенні різних нечітких безлічей.



Малюнок 6.4 Формування нечітких безлічей

Визначимо значення нечіткої безлічі «*немовля*»:

$$"\textit{немовля}" = \left\{ \frac{0,5}{1} + \frac{1}{0,9} + \frac{2}{0,8} + \frac{3}{0,7} + \frac{4}{0,6} + \frac{5}{0,3} + \frac{10}{0,1} \right\}$$

Основні операції з нечіткими безлічами

Позначимо через А і В нечіткі підмножини безлічі Х.

Тоді нечіткі безлічі А і В рівні, якщо:

А=У, якщо $\forall x \in X, \mu_A(x) = \mu_B(x)$

Безліч А міститься в У, якщо:

$$\forall x \in X, \mu_A(x) \le \mu_B(x)$$

Нечіткі безлічі А і В доповнюють один одного, якщо:

$$\forall x \in X, \mu_B(x) = 1 - \mu_A(x)$$

Безліч з функцією приналежності $1 - \mu_A(x)$ є доповнення до безлічі А (позначається $\overline{A}$).

Перетинання двох нечітких безлічей А і В (позначається $A \cap B$):

$$\mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}, \forall x \in X$$

Об'єднання двох нечітких безлічей А і В (позначається $A \cup B$):

$$\mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}, \forall x \in X$$

Операції перетинання й об'єднання нечітких безлічей асоціативні і дистрибутивні.

Алгебрагічний добуток нечітких безлічей А і В (позначається AB)є нечітка безліч, для якого

$$\mu_{AB}(x) = \mu_A(x) \cdot \mu_B(x), \forall x \in X$$

Алгебрагічна сума нечітких безлічей А і В (позначається $A + B$) є нечітка безліч з функцією приналежності

$$\mu_{A+B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)$$

Диз'юнктивна сума нечітких безлічей А і В визначається через операції об'єднання і перетинання

$$A \Delta B = (A \cap \overline{B}) \cup (\overline{A} \cap B) = (A \setminus B) \cup (B \setminus A) \ ,$$

де вираження $A \setminus B = A \cap \overline{B})$ і $B \setminus A = B \cap \overline{A}$ представляють відповідні різниці безлічей А і В. У загальному випадку $A \setminus B \neq B \setminus A$.

Важливим поняттям теорії нечітких безлічей є нечітке відношення. Нечітким бінарним відношенням $P : X_1 \to X_2$ називається підмножина декартового добутку двох безлічей X_1 і X_2 :

$$P = \{(x_1, x_2) \mid \mu_P(x_1, x_2)\}, \forall x_1 \in X_1, \forall x_2 \in X_2 \ ,$$

де $\mu_P(x_1, x_2)$ - функція приналежності пари елементів (x_1, x_2) до Р.

У загальному випадку *n* – арне відношення *P* визначається на прямому добутку безлічей $X_1 \times X_2 \times ... \times X_n$ за допомогою формули:

$$P = \{(x_1, x_2, ..., x_n) \mid \mu_P(x_1, x_2, ..., x_n)\} \ ,$$

де $x_1, x_2, ..., x_n$ елементи безлічі $X_1, X_2, ..., X_n$.

Нехай безлічі *X* і *Y* складаються з елементів *X*={1.2}, *Y*={1.2,2.1,5}.Задамо нечітку бінарну безліч у виді матриці, елементами якої будуть значення $\mu_P(x_i, y_i)$:

$$P = \begin{bmatrix} 0.9 & 0.1 & 0 \\ 0.1 & 0.9 & 0 \end{bmatrix} \begin{matrix} x_1 = 1 \\ x_2 = 2 \end{matrix}$$

$$y_1 = 1.1, y_2 = 2.1, y_3 = 5$$

Зокрема, дана матриця може представляти нечітке відношення "*X* приблизно дорівнює *Y*".

Часто нечіткі відносини використовуються для представлення правил типу якщо A то B , де A і B нечіткі підмножини ($A \subset X_1, B \subset X_2$). Таке правило позначає $A \Rightarrow B$. Один зі способів завдання нечіткої безлічі складається у використанні формули декартового добутку безлічей А і В (що позначається $A \times B$):

$$P = A \times B = \{(x_1, x_2) \mid \mu_A(x_1) \wedge \mu_B(x_2)\}, \forall x_1 \in X_1, \forall x_2 \in X_2 ,$$

Де

$$\mu_A(x_1) \wedge \mu_B(x_2) = \min\{\mu_A(x_1), \mu_B(x_2)\} = \mu_P(x_1, x_2) ,$$

причому $(x_1, x_2) \in X_1 \times X_2$.

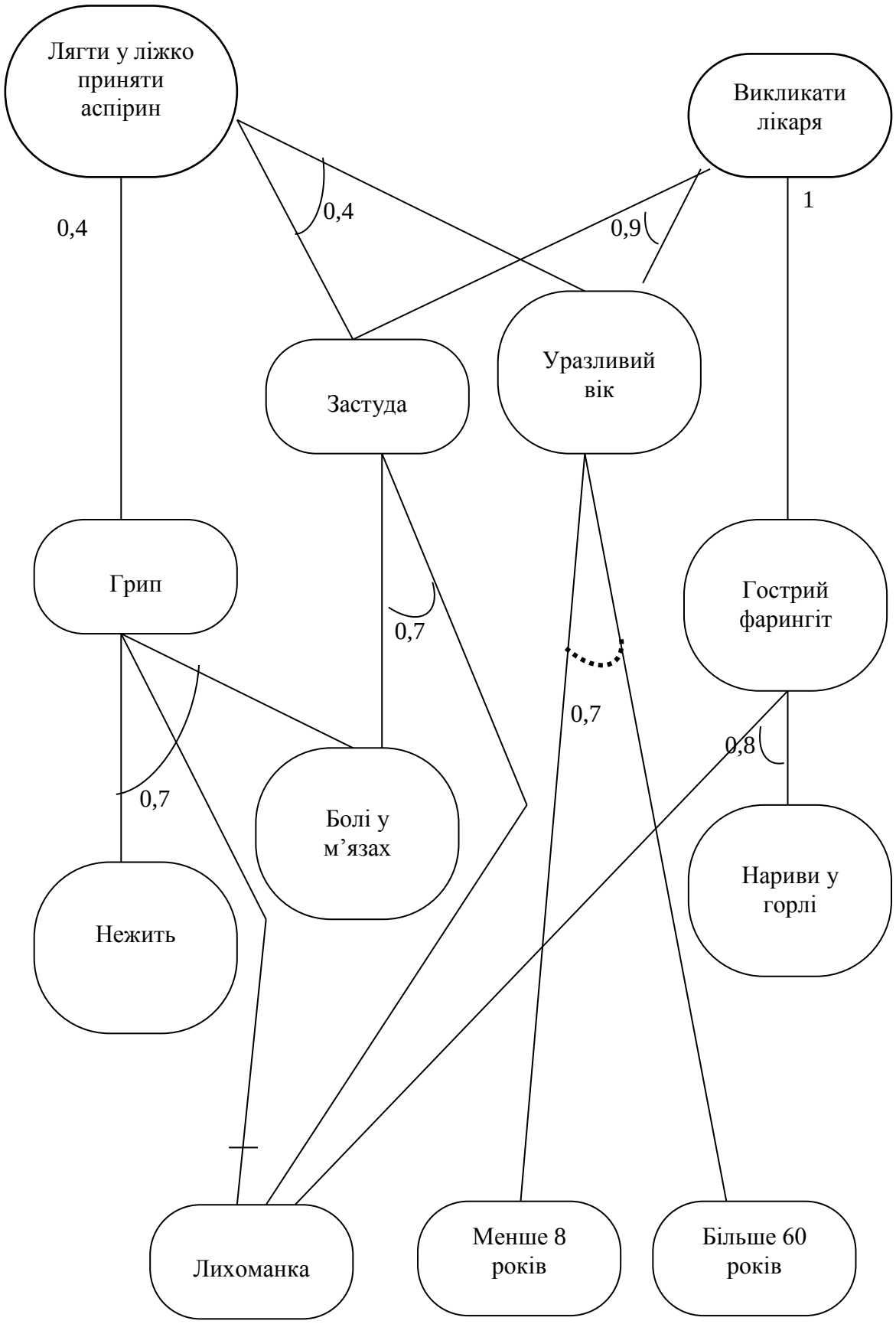
6.7. Багаторівневі міркування

Щоб уявити собі, що ж таке багаторівневі міркування, допустимо, що ви занедужали. У вас застуда, вірусна інфекція чи грип, і ви хотіли б знати, що варто робити. Число в правій частині кожного правила вказує коефіцієнт визначеності.

Якщо у вас грип і ви знаходитеся в уразливому віці, то викличте лікаря ct =.9. Якщо у вас гострий фарингіт, то викличте лікаря ct = 1.0. Якщо у вас застуда, то лягаєте в постіль і прийміть аспірин ct = .4. Якщо у вас грип і ви не знаходитеся в уразливому віці, то лягайте в постіль і прийміть аспірин ct=.4. Якщо у вас лихоманка і болят м'яза, то це грип ct = .7. Якщо у вас нежить, м'язові болі і немає лихоманки, то це застуда ct = .7. Якщо у вас у горлі нариви і є лихоманка, то це гострий фарингіт ct = .8. Якщо вам менше 8 чи більше 60 років, то ви в уразливому віці ct = .7.

Тепер потрібно переглянути правила й у залежності від конкретних симптомів захворювання вирішити, чи звернутися до лікаря чи досить лягти в постіль і прийняти аспірин. Можна використовувати комбінацію правил для визначення коефіцієнта визначеності, що відповідає кожному з можливих результатів, а потім вибрати той, котрий має найбільший коефіцієнт визначеності. Будь-яка система

правил може бути відображена графічно (Малюнок 6.5). Вона називається *мережею висновку*.

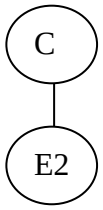


Малюнок 6.5. Медичні правила в мережі висновку

Приклад графічного представлення правил (Малюнок 6.6).

Проста імплікація:

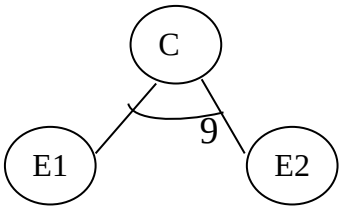
Якщо E то C ct = 0.8



8

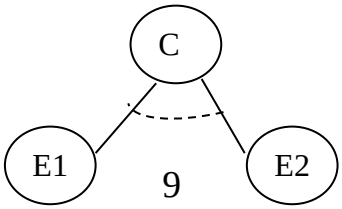
Імплікація AND:

Якщо (E1 and E2) то C ct=0.9



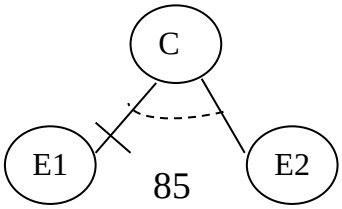
Імплікація OR:

Якщо (E1 or E2) то C ct=0.9



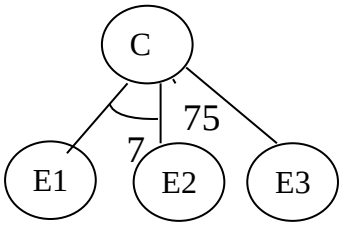
Імплікація з запереченням NOT:

Якщо ((not E1) or E2) то C ct=0.85



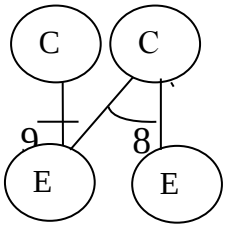
Кілька правил у підтримку одного висновку:

Якщо (E1 and E2) or E2) то C ct=0.7
Якщо (E3) то C ct=0.75



Одне свідчення, що використовується у двох правилах:

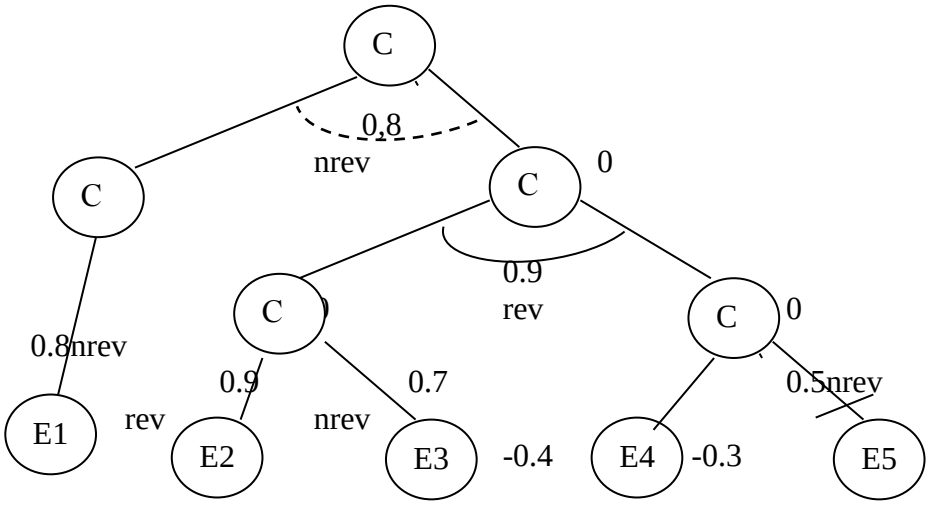
Якщо (not E1) то C1 ct=0.9
Якщо (E1 and E2) то C2 ct=0.8



Малюнок 6.6 Графічне представлення правил

6.8. Процес поширення в мережі

Розглянемо приклад, що ілюструє поширення коефіцієнтів визначеності в мережі (Малюнок 6.7).



Малюнок 6.7. Приклад мережі висновку для проведення міркувань з заданими початковими умовами

Вузли в основі дерева представляють умови з зовнішнього світу, про які система повинна задавати питання. Внутрішні вузли відображають висновок. Коефіцієнти визначеності внутрішніх вузлів цілком залежать від процесу міркування, правил імплікації і свідчень, отриманих із зовнішнього світу шляхом запитів. Число праворуч від кожного вузла відповідає коефіцієнту визначеності. До початку міркувань нічого не відомо про внутрішні вузли, тому вони усі мають коефіцієнти визначеності, рівні нулю.

Поруч з коефіцієнтом визначеності імплікації rev (оборотне) чи nrev (необратимо), що позначає, чи буде імплікація використовуватися як оборотне чи як необоротне правило. Обчислення коефіцієнта визначеності посилки може зажадати виконання декількох кроків: можуть додаватися "І", "ЧИ", "НЕ". У кожніму конкретному випадку, поки не буде закінчена вся ця попередня робота, не можна з упевненістю сказати, чи застосовне правило.

Мережа висновку (Малюнок 6.7) припускає наступні правила:

Якщо (e1), то (c1) ct = .8 (nrev)

Якщо (e2), то (c2) ct = .9 (rev)

Якщо (e3), то (c2) ct = .7 (rev)

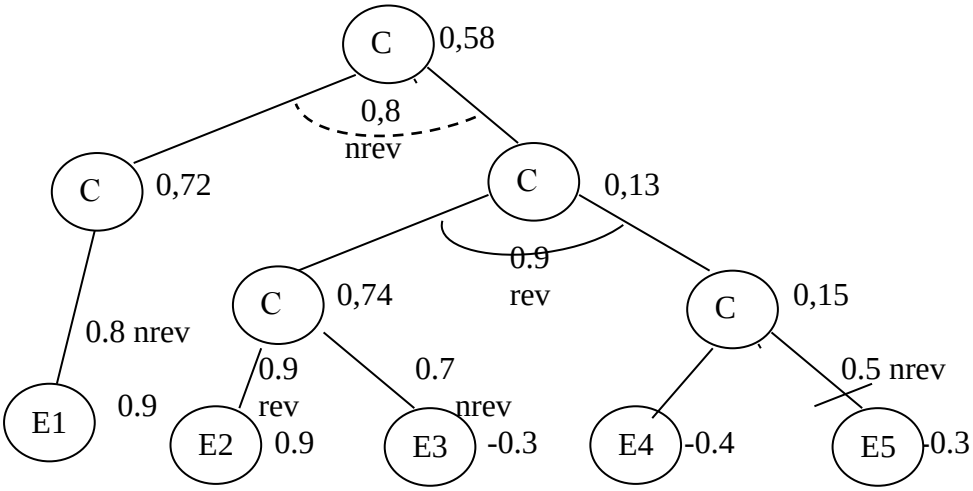
Якщо (e4), то (c3) ct = .6 (nrev)

Якщо (ні e5), то (c3) ct = .5 (nrev)

Якщо (c2 і c3), то (c1) ct = .9 (rev)

Якщо (c1 чи c4), то (c5) ct = .8 (nrev)

Нижче (Малюнок 6.8) показан результат усіх міркувань, проведених у мережі, що використовує придатне свідчення.



Малюнок 6.8. Приклад мережі висновку з обчисленими коефіцієнтами

Коефіцієнт визначеності C1 може бути обчислений у такий спосіб:

ct(висновок C1) = .8 * .9 = .72.

Це проста необоротна імплікація, але оскільки коефіцієнт визначеності посилки позитивний, правило можна застосовувати.

При обчисленні коефіцієнта визначеності C2 два правила використовуються без обмежень, тому що вони оборотні. Правило ліворуч дасть оцінку коефіцієнта визначеності C2:

ct(висновок C2) = .9 * .9 = .81.

Правило праворуч дасть іншу оцінку:

ct(висновок C2) = -.3 * .7 = -.21.

Тут два підтримуючих правила, що дають оцінку коефіцієнта визначеності з протилежними знаками, тому для остаточної відповіді об'єднаємо ці оцінки:

ct(висновок С2) = (.81+ (-.21))/(1 —.21) = .74.

Для С3 знову два правила. Правило, зв'язане з лівим піддеревом, не застосовується, так- як воно необратимо, і коефіцієнт визначеності посилки негативний. Правило, зв'язане з правим піддеревом, є проста імплікація. Вона необоротна і містить негативну посилку. Правило затверджує:

Якщо (ні Е5) то (С3) ct = .5 (nrev)

Коефіцієнт визначеності Е5 дорівнює -.3. Тому що він негативний, то коефіцієнт визначеності посилки в правилі дорівнює .3. Правило необоротне, але посилка знаходиться в припустимому інтервалі. Використовуючи процедуру, призначену для простої імплікації, знайдемо для С3:

ct(висновок С3) = .3 * .5 = .15.

Імплікація, що підтримує С4, включає кон’юнкцію посилок. Коефіцієнт визначеності посилки дорівнює:

ct (свідчення) = min(.15, .74) =.15.

Оскільки правило оборотне, можна використовувати посилку в будь-якому інтервалі визначеності. Використовуючи цей результат, обчислимо коефіцієнт визначеності для С4:

ct (висновок С4) =.15* .9 =.13.

Тепер пійшли нагору по дереву до того місця, де можна судити про вузол верхнього рівня. Тут задіяне одне правило, у якому посилки розділені за допомогою ЧИ, тому:

ct (свідчення) = мах (.72, .13) = .72.

Правило необоротне, але коефіцієнт визначеності посилки позитивний, так що можемо рухатися далі.

Остання ланка в нашому ланцюзі міркуванні - коефіцієнт визначеності для вузла вищого рівня - обчислюється по формулі:

ct (висновок с5) = .72 * .8 =.58.

Звичайно мережі висновку вимагаються в ситуації, коли при наявності декількох конкуруючих гіпотез експертна система намагається зробити вибір і породити інформацію. Наприклад, у медичних системах мережа вибору може використовуватися при встановленні причин хвороб у пацієнта: чи апендицит це, рак, яка-небудь інфекція, що викликала запалення лімфатичних вузлів, чи, можливо, простий розлад шлунка.

Програма ілюструє гарний спосіб представлення зв'язків і імплікацій у типовій мережі висновку. Крім того, для усіх вузлів, названих тут конкуруючими гіпотезами, програма може одержувати інформацію доцільним образом і міркувати, щоб зробити вибір між гіпотезами верхнього рівня на основі обчислених визначеностей.

Усі правила оборотні. Вони можуть бути використані для всіх можливих значень коефіцієнта визначеності посилки.

Зв'язок між вузлами представляє основні кроки міркувань. Включаємо їх у програму за допомогою фактів імплікації. Наприклад, у c1 розташований вузол I. Його структура у формі правила виглядає так:

Якщо ((ні e1) і e2), то (cl) ct = .9 (rev)

Предикат мовою Prolog може бути таким:

imp(a, r, c1, neg, el , pos, e2, 0.9).

де a – тип зв'язку I (AND) чи про – ЧИ (OR) чи s – простий зв'язок; r – правило оборотне (rev); c1 – ім'я вузла; neg (чи pos) - перед кожним вузлом указує шаблон заперечення в правилі, якщо такий мається. Коефіцієнт визначеності даної конкретної імплікації дорівнює .9.

Всі інші зрозумілі програмі імплікації можна представити за допомогою тих же восьми аргументів фактів імплікації. Проста імплікація представляється так:

lmp(s,r,c3,pos,e5, dummy, dummy,0.6).

Тут записано, що вузол c3 підтримується зазначеною імплікацією, а e5 - посилка. Посилка не заперечується, тому правило зворотне. Аргументи 6 і 7 (dummy,dummy) будуть завжди присутні в простих імплікаціях, так що можна використовувати ту саму форму для представлення всіх імплікацій, про які відомо програмі.

Контрольні питання

1. Допустимо, у задачі на всі питання отримані відповіді ТАК чи НІ. Зосередимо увагу на двох конкретних питаннях, що назвемо "питання А" і "питання В". Споживачі мають тенденцію відповідати на ці питання однаково, тобто ті, хто відповів ТАК на питання А, імовірно, так само відповідять і на питання В, а ті, хто відповів НІ на питання А, імовірно, так само відповідять і на питання В.

Нехай p(A) - імовірність відповіді ТАК на питання А. Відомо, що p(A) = .6 і відповіді на обидва питання значно корелюють один з одним, тому:

p(У | А)=.9; p(А | У)=.8; p (ні В | ні А) =.85.

Обчислить наступні імовірності: p(У); p(не А | не В); p(А і В).

Яка імовірність того, що в будь-якому іспиті відповіді на питання А і В будуть однакові?

2. Вкажіть, чи буде зворотним кожне з наступних правил:

Якщо команда робить гарні кидки, то вона добре грає.

Якщо людина жива, то в ній тече кров.

Якщо були ранні заморозки, то врожай персиків буде поганим.

3. Дано: Якщо (ні X), то C; і ct(X) ~ .8. ct(імплікація) = .7(nrev)

Який коефіцієнт визначеності посилки для цього правила? Яка визначеність висновку?

7. Нейросмережева технологія

7.1. Особливості нейромереж

Головне достоїнство [57] нейромереж у тім, що вони дають у руки користувачу деякий *універсальний нелінійний елемент* із можливістю широкої зміни і настроювання його характеристик. Розташовуючи свого роду конструктором з таких елементів і з'єднуючи їх у мережу, користувач, з одного боку, одержує можливість широкої зміни її характеристик, а з іншого боку- може особливо не замислюватися над процесами, що відбуваються в цій мережі. Їм заздалегідь гарантована цілеспрямованість і оптимальність, що приводять в остаточному підсумку до досить прийнятного результату. Поява нейромереж вкладається в загальну для всієї інформаційної індустрії тенденцію- перехід від деталей до великоблочного будівництва (Case-системи, об’єктно-оріентованні технології і т.п.).

Набір нелінійних адаптивних елементів дозволяє моделювати будь-яке нелінійне перетворення і набудовувати його на різні задачі автоматично шляхом зміни параметрів у процесі навчання. Причому останнім часом спостерігається тенденція використовувати для настроювання не емпірично знайдені прийоми (типу правила Хебба, зворотнього поширення помилки і т.п.), а універсальні і добре відпрацьовані математичні методи пошуку экстремума цільової функції в просторі параметрів. Це стосується і вибору цільової функції: перехід від часток емпірично знайдених форм (аналог енергії в мережах Хопфілда, сумарна квадратична помилка в методі зворотнього поширення) до більш загального.

Місце нейронних мереж у системах обробки інформації можна вказати за аналогією зі структурою людської психіки: воно відповідає нижчому інтуїтивному рівню реакцій, коли потрібно швидка відповідь на досить стандартну ситуацію. Якщо відповідь не знайдена чи система сумнівається в його вірногідності, то керування передається більш високому логічному рівню.

Йому відповідає експертна система, що розташовує широкою базою знань і здатна робити більш обґрунтовані висновки.

Нейронні мережі здатні вирішувати такі задачі, як розпізнавання образів, виділення сигналу на тлі шуму, виправлення помилок, керування складною адаптивною системою керування при неможливості формалізувати експертні знання чи при відсутності таких і т.п. Усе це вже знаходить широке практичне застосування (деякі приклади приведені нижче). Нейромережа може запам'ятовувати дії досвідченого оператора, що керує складною системою, а потім відтворювати їх, виявляючи необхідну гнучкість, змінюючи зразки поводження і вибираючи серед них той, котрий найбільш близький і адекватний поточної ситуації. При цьому немає необхідності алгоритмиризувати діяльність оператора, щоб потім на її основі будувати програму керування: система схоплює форми поводження цілісно як нерозкладне ціле і створює для їхньої реалізації відповідні структури.

У загальному випадку в поводженні такої системи [27] варто розрізняти три задачі:

– навчання і запам'ятовування поведінкових зразків (еталонів), що задаються зовнішніми умовами. При цьому відбуваються утворення і модифікація зв'язків між елементами;

– розпізнавання зовнішньої ситуації, віднесення її до одного з запом’ятованих еталонів, вибір відповідного поведінкового зразка;

– реалізація обраного еталона поводження, підтримка еталонних значень змінних, повернення до них після збурювань, виправлення помилок і нейтралізація перешкод, створюваних зовнішнім середовищем.

В окремому випадку третя задача може бути відсутня і робота системи може завершуватися розпізнаванням ситуації.

7.2. Властивості нейрона

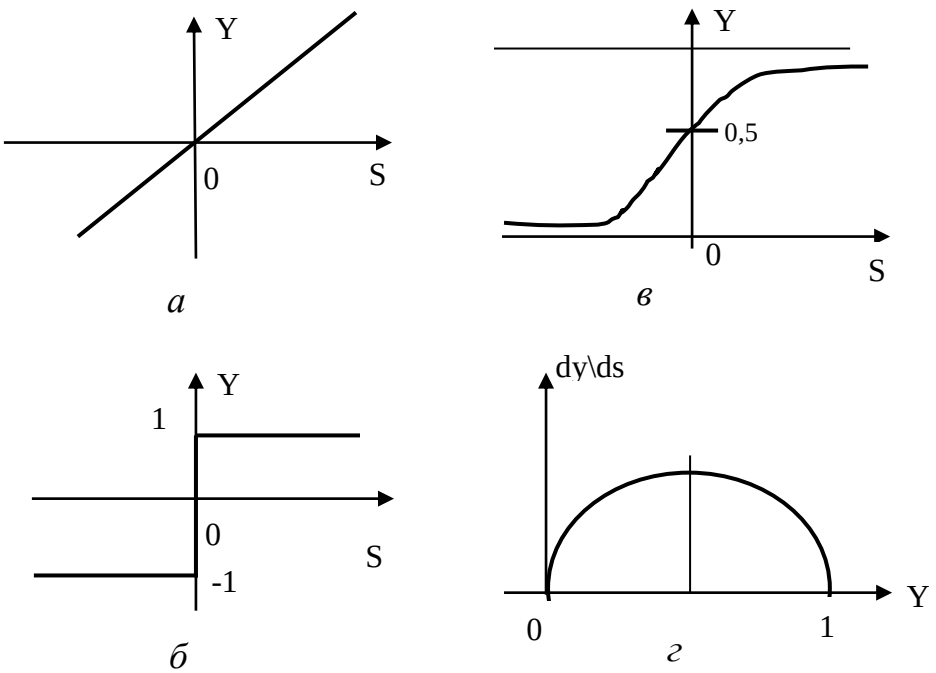
З конструктивної точки зору *нейрон,* що є основним елементом нейромережі, це пристрій для одержання нелінійної функції декількох змінних X_i з можливістю настроювання його параметрів C_j у досить широкому діапазоні. Однак традиційно нейрон описується в термінах, запозичених з фізіології. Згідно з цими представленнями нейрон має один вихід *Sj* і кілька входів (сінапсів), на які надходять зовнішні впливи *Xj*, (від рецепторів і інших нейронів). Він множить вхідний вплив на ваговий коефіцієнт C_{ij} (провідність сінапса) і підсумовує зважені входи:

$$s_i = \sum_i c_{ij} x_i + c_{0j} \tag{7.1}$$

Вихідна величина y_i є деякою функцією від цієї суми: $y_i=f(Sj)$. Її називають *функцією активації чи передатною функцією.* Вид цієї функції є найважливішою характеристикою нейрона. У найпростішому випадку - це *лінійна* залежність (Малюнок 7.1, а)

$$y_i = ks_j = k(\sum c_{ij} x_i + c_{0j})$$

Така залежність використовувалася в перших моделях персептрона. Незважаючи на ряд первісних успіхів, теоретичний аналіз можливостей персептрона, проведений М.Мінським і С.Пейпертом [49], показав, що персептрон не є універсальним пристроєм для розпізнавання і, зокрема, принципово нездатний вирішити цілий ряд дуже простих задач. Причиною цього є саме лінійний характер активаційної функції.



Малюнок 7.1 Функції активації нейронной мережі: *а* – лінійна функція;
б- східчаста функція; *в*- сигмоідальная функція;
г- похідна від сигмоідальної функції

Ще використовувалася с*ходчаста* функція активації: якщо сума *Sj* вище деякої межі значення c_{0j}, то вихід y_j дорівнює одиниці, у противному випадку- мінус одиниці (чи нулю). Формально це можна описати за допомогою наступної залежності (Малюнок 7.1, б):

$$y_i = \mathrm{sgn}(s_j) = \mathrm{sgn}\sum c_{ij}x_i + c_{0j})$$

В цей час як активаційну функцію частіше використовують близьку до сходчастої, але більш гладку залежність, що називають *сігмоїдальной, чи логістичной,* функцією (Малюнок 7.1, в). Звичайно вона описується наступним виразом:

$$y = 1/(1+e^{-ks})$$

Зустрічаються й інші вирази, наприклад,

$$y = s/(1 + k \mid s \mid) \ ,$$

де |s| - абсолютна величина s, k> 0.

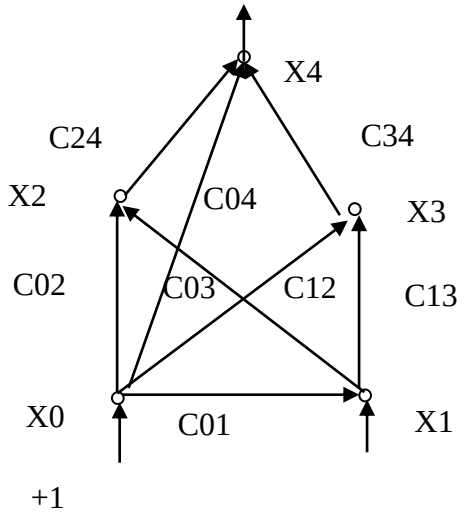
Параметр *k* задає крутість залежності *y* від s: чим більше *k*, тим ближче сігмоіда до граничної функції; чим менше *k,* тим ближче вона до лінійного. Таким чином, сігмоіда є деяким компромісом між лінійною і східчастою функцією, що зберігає достоїнства обох. Подібно східчастої функції, вона нелінійна, і це важливо, оскільки тільки нелінійні функції дозволяють вичліняти у просторі ознак безлічі складної форми, у тому числі невипуклі і незв'язні. Але в той же час сігмоіда на відміну від східчастої функції переходить від одного значення до іншого без розриву, як це має місце й у лінійній функції. Ця обставина виявляється надзвичайно важливим при пошуку экстремума цільової функції в просторі нейроних параметрів, у цьому випадку залежність цільової функції від параметрів також виявляється гладкою, і в кожній крапці простору може бути обчислений градієнт цільової функції, що вказує напрямок пошуку экстремума.

Похідна від сігмоідальної функції, що характеризує силу зв'язку між *s* і *в,* також має простий вид:

$$dy / ds = ky(1 - y) \qquad (7.2)$$

Ця величина перетворюється в нуль на границях діапазону зміни *в* при *y*=0 і *в=1* і досягає максимуму в середині діапазону, тобто зв'язок між змінними найбільш сильний в середині діапазону і слабший по краям (Малюнок 7.1, *г*).**Ошибка! Источник ссылки не найден.**Нейрони організуються в мережу (Малюнок 7.2) за рахунок того, що вихід *i*-го нейрона (y_i) з'єднується з одним із входів (x_i) іншого *j-го* нейрона. При цьому вихідна змінна y_t ототожнюється з вхідною змінною x_i. Тому надалі будемо використовувати два значення у залежності від того, чи розглядається дана *i-я* змінна як вхідна чи як вихідна. Ваговий коефіцієнт c_{ij} ("сінаптічна вага") характеризує знак і силу зв'язку між змінними x_i і y_i. Можливий і

зворотний зв'язок, при якому вихід *j-го* нейрона з'єднується з *j-м* входом *j-го* нейрона. У загальному випадку коефіцієнт зв'язку c_{ij}, не обов'язково дорівнює c_{ji}.



Малюнок 7.2 Приклад нейронной мережі

Найважливішою властивістю нейрона є його *пластичність* - можливість змінювати параметри в процесі навчання:. У ранніх роботах по нейромережам звичайно розрізняли два типи пластичності:сінаптічну (зміна c_{ij}) і нейронну (зміна висоти порога нейрона z_{0j}) У даний час межу пластичності, звичайно зводять до сінаптіческой за допомогою наступного прийому. До числа входів *j-го* нейрона додають ще одним фіктивний x_0, не зв'язаний ні з яким реальним рецептором (**Ошибка! Источник ссылки не найден.**).На цей вхід подають постійний сигнал, рівний +1. Ваговий коефіцієнт цього входу z_{0j} модифікують у процесі навчання за загальними правилами. Модифікація цього коефіцієнта рівносильна зсуву порога нейрона.

Ще в 1949 р. Д. Хеббом [28] було запропоновано природне *правило модифікації вагових коефіцієнтів: якщо два нейрони збуджуються разом, то сила зв'язку між ними зростає; якщо вони збуджуються порізно, те сила зв'язку між ними зменшується.* Правило виявилося настільки вдалим, що дотепер використовується в різних моделях нейронних систем. Формальніше це правило може бути описане в такий спосіб. Нехай час навчання розбитий на такти й у *k*- м такті дві змінні

нейромережі (стани двох нейронів) мали значення x^k_i , і x^k_j. Тоді вага зв'язку між змінними зростає на величину

$$\Delta c^k_{ij} = x^k_i x^k_j$$

У випадку двійкових змінних збільшення дорівнює або +1 (при збігу знаків x^k_i і x^k_j), або -1 (коли знаки різні). Якщо початкова вага зв'язку дорівнює нулю, то вага зв'язку до *k-го* такту дорівнює:

$$cp_{ij} = \sum_1^p x^k_i x^k_j$$

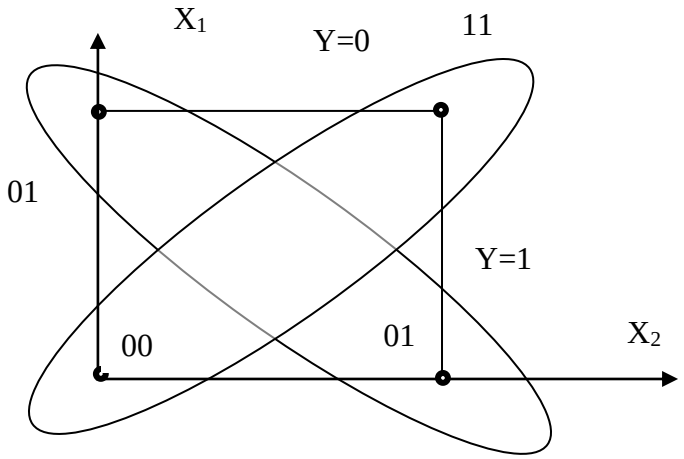
де x^k_i , x^k_j - стану двох нейронів у *k-м* такті; *p* - число тактів навчання.

7.3. Використання нелінійних елементів

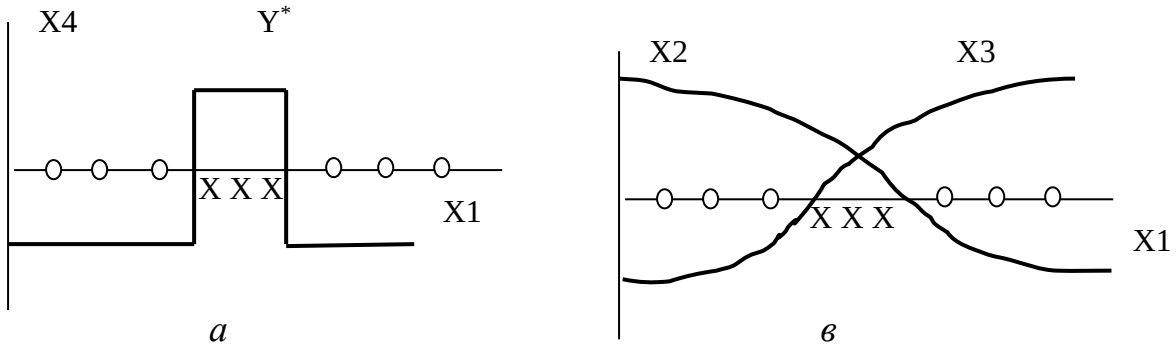
Один із самих несподіваних результатів аналізу І.Мінського і С.Пейперта полягав у тому, що персептрон, побудований на лінійних функціях активації, не може відтворити таку просту логічну функцію, що як виключає ЧИ (XOR). Це функція двох аргументів $y(x_1, x_2)$ усього можливі чотири комбінації значень аргументів: 00, 01, 10, 11. У просторі ознак вони розташовані по кутах одиничного квадрата (Малюнок 7.3).Функція $в = (x, \text{XOR } x_2)$ дорівнює одиниці, коли дорівнює одиниці один з аргументів, але не обоє відразу. Таким чином, безліч крапок (01, 10) відноситься до класу, де $в = 1$, а безліч (00, 11) - до класу $в = 0$. У просторі ознак елементи цих класів лежать на протилежних кутах квадрата і ніяка лінійна функція $в(x_1, x_2)$ від цих ознак не здатна розділити ці два класи. У цьому випадку не допомагає і використання другого шару нейронів, тому що добуток двох лінійних перетворень знову дає лінійне перетворення, що володіє тими ж недоліками. Виходом є використання нелінійних елементів.

Для ілюстрації цього висновку розглянемо простий приклад, коли простір ознак є одномірним. Навіть у цьому випадку легко побудувати задачу, що не може бути вирішена з використанням тільки лінійних функцій активації.

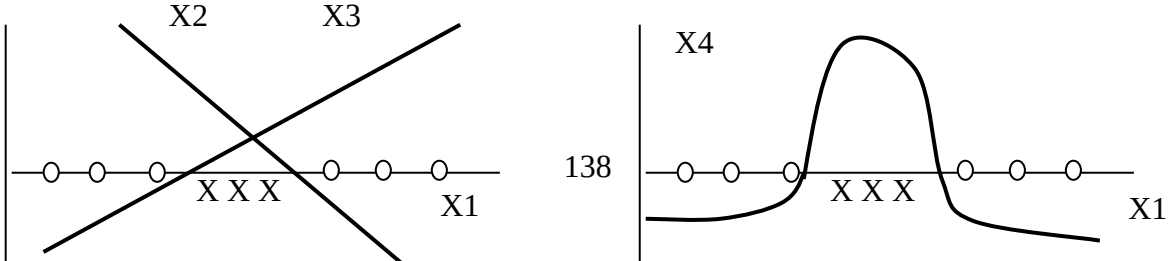
Нехай в одномірному просторі ознак *x* елементи одного класу (нулики) розташовані навколо елементів іншого класу (хрестиків) (Малюнок 7.4), т.б. безліч нуликів є незв'язними. Необхідно побудувати таку функцію *f(x\)*, що приймала би позитивне значення на хрестиках і негативне- на нуликах. Це дозволить розділити весь простір *x* на *три* області і потім крайні області (негативні значення $f(x_1)$) об'єднати в один клас, а середню (позитивні значення *а)* віднести до іншого. Бажана (ідеальна) залежність *y'(x),* що вирішує цю задачу– Малюнок 7.4, *а*. Як видно з рисунка , вона є нелінійною: потрібна крива щонайменше другого порядку,



Малюнок 7.3 Простір ознак для функції *в*=x_1 хог x_2



а *в*



Малюнок 7.4. Використання нелінійних функцій активації:
а - ідеальна залежність; *б* - лінійні залежності;
в - сігмоідальні залежності; *г* - рішення задачі за допомогою нейронной мережі із сігмоідальними залежностями

щоб вона перетнула ось *x* двічі і розділила її на три частини. З малюнка 7.4, *б* випливає, що ніякі лінійні залежності чи їхні комбінації (в одномірному просторі це просто прямі) не дозволяють вирішити цю задачу: будь-яка комбінація прямих дає знову пряму, а вона перетинає весь *x* тільки в одній крапці і поділяє весь простір *x* тільки на *дві* області. Інша справа- нелінійні, наприклад східчасті чи сігмоідальні, функції (Малюнок 7.4, *в)*.

Для реалізації необхідної залежності необхідна мережа, що складається з п'яти нейронів (Малюнок 7.2).Нейрон x_1 є рецептором, він сприймає значення "ознака x_1" і просто транслює його далі. Нейрон x_0 є "псевдонейроном"- його задача створювати постійний сигнал +1, що, помножений на ваги c_{0i}, формує граничні значення для інших нейронів. Нейрони x_2 і x_3 -основні робітники нейрони мережі. Їхніми активаційними функціями є сігмоіди (Малюнок 7.4, *в)*. Нахил характеристики- позитивний для x_3 чи негативний для x_3- задається ваговими коефіцієнтами c_{13} і c_{12}, а положення на осі *x*- порогами c_{03} і c_{02} відповідно:

$$x_2 = f(c_{12}x_1 + c_{02}) \qquad x_3 = f(c_{13}x_1 + c_{03})$$

Нарешті, вихідний нейрон x_4 просто підсумовує ці сігмоіди- з якимись порогами і вагами:

$$x_4 = c_{24}x_2 + c_{34}x_3 + c_{04}$$

У результаті виходить залежність виходу мережі x_4 від входу x_1, що вирішує задачу поділу хрестиків і нуликів: вона позитивна для хрестиків і негативна для нуликів (Малюнок 7.4, *г)*.

Одна з тенденцій у розвитку нейронних мереж складається в переході до більш гнучких і універсальних нелінійних функцій. Суть тенденції може бути зрозуміла з наступного міркування: *метою* навчання звичайно є виділення *областей*, займаних різними класами. *Засобом* же є проведення меж, оскільки межевий елемент- лінійний чи нелінійний- визначає саме межу в просторі ознак. І тільки на наступному рівні

ієрархії за допомогою межі виділяються області. Такий шлях незручний, особливо якщо область, займана класом, має складну форму, є багатосв'язной (як у випадку що виключає ЧИ). Очевидно, буде краще (принаймні в деяких випадках), якщо з елементом зв'язується не межа, а відразу деяка стандартна елементарна область, що може служити базисом для побудови більш складних областей. Наприклад, можна описувати нейрон сходчатою активаційною функцією *в(х),* позитивної в деякой області простору ознак і негативної- у всіх інших областях цього простору (Малюнок 7.4, *а)*. Параметрами, що набудовуються, при цьому можуть бути розміри області і її положення в просторі ознак. Тоді задачу про поділ хрестиків і нуликів можна було вирішити за допомогою єдиного нейрона. Підсумовуючи виходи декількох таких нейронів, можна легко виділити область самої складної форми. Східчасту функцію при цьому можна, звичайно, згладити (наприклад, як на малюнку 7.4, *г)*, щоб мати можливість використовувати градиентні методи пошуку экстремума.

З класичних методів розпізнавання найбільш близький до цього відомий метод потенційних функцій. Зараз використовують нейромережі саме такого роду функції (радіальні базисні функції, р-функції і т.т.). Так, сферична радіальна базисна функція *і-го* нейрона може задаватися виразом, аналогічним виразу для нормального розподілу. Комбінація елементів такого чи подібного типу здатна апроксимувати будь-яку нелінійну залежність і, отже, виділити в просторі ознак області самої складної форми- невипуклі, багатосв'язні і т.т.

У цілому архітектура нейромережі може бути задана матрицею вагових коефіцієнтів c_{ij} зв'язків, що характеризують силу, між елементами мережі. У загальному випадку всі елементи зв'язані з усіма, але матриця зв'язків несиметрична, $c_{ij} \neq c_{ji}$. Деякі коефіцієнти зв'язків можуть залишатися вільними, незаданими і тоді можливо їхня зміна- навчання мережі.

Таким чином, накладаючи умови на значення c_{ij}, визначається конфігурація мережі. При цьому з безлічі можливих конфігурацій одержали поширення і достатнє добре досліджені лише деякі. До числа найважливіших відносяться дві конфігурації:

1) одношарова мережа Хопфілда;

2) тришарова мережа з проміжним шаром "схованих" нейронів.

7.4. Мережа Хопфілда

У 1982 р. з'явилася робота Дж. Хопфілда [9], що викликала лавину теоретичних і експериментальних досліджень і оживила інтерес, що вгасав, до нейронних мереж. Несподіваний успіх роботи підрозуміється під використанням простого й ефективного математичного апарата, що дозволив побачити нові грані проблеми й одержати ряд нових результатів чисто теоретичним шляхом. Подібність цієї мережі з деякими добре дослідженими фізичними моделями (модель Ізінга, спінові стекла й ін.) дозволило використовувати для аналізу готовий і добре відпрацьований апарат статистичної термодинаміки [57].

Мережа Хопфілда виходить, якщо накласти на ваги зв'язків наступні умови:

1) всі елементи зв'язані з усіма;

2) $c_{ij} = c_{ji}$ - прямі і зворотні зв'язки симетричні;

3) $c_{ij} = 0$- діагональні елементи матриці зв'язків дорівнюють нулю. Остання умова звичайна (хоча і не завжди) додається, щоб виключити безпосередній зворотний зв'язок з виходу нейрона на вхід.

Одне з достоїнств симетричної квадратної матриці зв'язків, характерної для мережі Хопфілда, полягає в тому, що поводження мережі можна описати через прагнення до мінімуму простої цільової функції

$$E = -\sum_{i \neq j} c_{ij} x_i x_j = \min$$

Звичайно E інтерпретується як деяка узагальнена енергія. У моделі Хопфілда коефіцієнти зв'язків можуть приймати будь-які значення, як позитивні, так і негативні; ці значення не задані раз і назавжди, а міняються в процесі навчання.

Поводження системи в просторі станів нагадує рух кульки, що прагне скотитися в крапку мінімуму деякого потенційного рельєфу. Характер рельєфу визначається видом цільової функції E и формується в процесі навчання мережі. Навчання виробляється шляхом демонстрації еталонних образів, що мережа повинна

запам'ятовувати, зберігати і потім відтворювати (дізнаватися). Алгоритм навчання (формування вагових коефіцієнтів c_{ij}) ґрунтується на правилі Хебба.

Чудова властивість такої мережі полягає в тому, що та сама мережа з тими самими вагами зв'язків може зберігати і відтворювати кілька різних еталонів.

Хоча мережі Хопфілда набули застосування на практиці (часто як складова частина більш складних систем), однак їм властиві визначені недоліки, що обмежують можливості їхнього застосування:

• модель Хопфілда припускає симетрію зв'язків між елементами; без цієї умови поняття енергії не може бути введено, і ця проста фізична метафора, як модель багато в чому зобов'язана своїм успіхом, перестає працювати;

• умовність поняття енергії змушує відноситися до нього з обережністю. Це тільки метафора, красива, але спотворююча суть процесів, що відбуваються. Нейронна (і її прототип- нервова) мережа є пристроєм для мінімізації енергії; цей пристрій для запам'ятовування й обробки інформації.

7.5. Багатошарові мережи

Мережа Хопфилда підтримує безліч зайвих, неефективних зв'язків, власне кажучи дублюючих один одного. У реальних нервових системах підтримка таких зв'язків вимагає визначених витрат і тому невигідно. Тому в ході еволюції нервової системи відбувалося звільнення від частини зв'язків за рахунок централізації системи зв'язків. Подібна централізація є загальносистемною закономірністю і спостерігається в багатьох системах- біологічних, технічних, соціальних. Наприклад, перші телефонні мережі безпосередньо зв'язували абонентів один з одним. Однак з ростом числа абонентів число зв'язків N росло приблизно пропорційно квадрату числа абонентів n:

$$N = n(n-1)/2$$

і мережі швидко ускладнювалися. Тоді були введені центральні телефонні станції, так що кожен абонент тепер з'єднувався безпосередньо тільки з телефонною станцією і вже через неї- з іншими абонентами. Число зв'язків різко зменшилося:

$$N = n,$$

а з ними зменшилися і витрати на їхню підтримку.

Подібну еволюцію проробила і нервова система тварин- від дифузійної в найпростіших до центральної нервової системи і головного мозку у вищих ссавців. Тому зв'язок багатьох елементів з одним, центральним варто вважати більш високим принципом організації, чим зв'язок "усіх із усіма". Безліч центральних елементів утворить новий рівень або шар, для якого, у свою чергу, справедливий той же принцип організації. Так виникає багатошарова ієрархічна система зв'язків. Схильність до такої організації виявляють і системи обробки інформації, як природні, так і штучні. Їх можна знайти вже в традиційних системах розпізнавання образів - у виді ієрархічної організації системи ознак, коли з простих ознак будуються більш складні, а з них уже - фрагменти образів і далі - самі образи.

Як приклад такого підходу розглянемо метод групового обліку аргументів, запропонований А.Г. Івахненко [38], або його аналог - метод δ-функцій [37], основна ідея яких полягає в наступному.

Будь-яку нелінійну функцію n ознак x_i , що задає бажане відображення вхідного простору у вихідне, можна апроксимувати за допомогою полінома:

$$y = a_0 + \sum_i a_i x_i + \sum_i \sum_j a_{ij} x_i x_j + \ldots + \sum_i \sum_j \ldots \sum_n a_{ij\ldots n} x_i x_{j\ldots} x_n$$

Коефіцієнти полінома повинні бути підібрані так, щоб забезпечити бажану залежність y від x і розпізнавання з мінімальною помилкою. Для точного відшукання коефіцієнтів можна використовувати систему нормальних рівнянь Гаусса. Однак, при скільки-небудь високому ступені полінома і при достатній кількості ознак розміри матриць цієї системи рівнянь ростуть катастрофічно. Так, при десятьох ознаках матриця містить $2*10^5*2*10^5$ елементів. Цей процес виявляє

себе і при використанні адаптивних методів відшукання коефіцієнтів поліномів: час навчання виявляється неприйнятно великим.

У математиці давно вже знайдений вихід з цієї ситуації. Він складається у використанні різних систем стандартних функцій, таких, як поліноми Чебышева, Эрмита, гармонійні функції й ін. Сутність підходу- в ієрархічній організації складних залежностей. Стандартні функції підбираються так, що вони, з одного боку, самі вже мають досить складні і цікаві властивості, з іншогобоку- їхній можна досить просто комбінувати для апроксимації ще більш складних функцій. Іншими словами, поліном будується не з найпростіших- статечних функцій, а з більш складних. При цьому виявляється, що неможливо запропонувати єдину систему стандартних функцій, придатну на усі випадки життя,- для кожної області додатків найбільш підходящої виявляється своя система стандартних функцій.

У методі групового обліку аргументів складний поліном заміняється декількома більш прості, враховуючі тільки деякі ознаки ("групи аргументів"). При цьому кожен спрощений поліном розглядається як самостійний незалежний класифікатор, коефіцієнти якого визначаються шляхом рішення системи нормальних рівнянь Гаусса малої розмірності (тобто точним методом). Після навчання відбирається трохи найкращих (у змісті результатів класифікації) поліномів, і їхні ліві частини y_j ("складні ознаки") використовуються як аргументи для побудови більш складного полінома. Практично використовувалися різні попарні об'єднання ознак, що давало можливість будувати як границі прямі і криві другого порядку.

Таким чином, ієрархічна організація ознак- загальний шлях, що забезпечує компроміс між бажаною точністю і прийнятними витратами на пошук або навчання. Можливість саме такої організації і надає користувачу нейронная мережа з її готовим набором стандартних нелінійних функцій. Якщо раніш як такий стандарт виступала гранична залежність, що дозволяє проводити границі в просторі ознак, то зараз як альтернативу застосовуються вже й інші стандартні набори (потенційні функції, радіальні базисні функції й ін.), що оперують не з границями, а безпосередньо з областями.

Необхідність ієрархії багато в чому і визначає структуру більшості сучасних нейронных мереж. Найважливішим нововведенням і головною відмінною рисою цієї структури є наявність проміжного шару (або декількох шарів) "схованих нейронів". Сховані елементи не є узкоспециализированными, вони не зв'язані жорстко ні з вхідними зразками, ні з вихідними реакціями, ця воля і додає нейросети незвичайну гнучкість, обчислювальну потужність і здатність адаптуватися до самих різних комбінацій входів і виходів. Наявність схованих елементів дозволяє нейросети виконувати дії, подібні з тими операціями по перетворенню і скороченню вихідних даних, що давно уже використовуються в багатомірній статистиці. От деякі очевидні аналоги функцій, виконуваних схованими елементами:

- "головні компоненти" у факторному аналізі;
- "координати" у багатомірному шкалировании;
- "дискриминантные функції" у дискриминантном аналізі.

Мережа зі схованими елементами може реорганізувати простір вхідних ознак у прості області, потім об'єднати їх у більш складні (неопуклі, незв'язні) і, нарешті, асоціювати їх з вихідними категоріями.

Іншими факторами, що визначають архітектуру нейросети, є умови зв'язку з зовнішнім середовищем: мережа повинна мати число вхідних елементів, рівне розмірності простору ознак; число вихідних- розмірності простору відповідей. Число проміжних (схованих) елементів визначається складністю задачі, необхідним обсягом пам'яті і припустимою помилкою розпізнавання.

7.6. Динаміка навчання

В механіці динаміка на відміну від статики і кінематики припускає наявність двох моментів: зміну змінних у часі й обумовленість цих змін силами. Ці два моменти мають сенс стосовно і до нейронным мереж. Сила тут не просто метафора. Їй можна дати точне визначення і кількісне вираження. Це робить її корисним інструментом для опису поводження системи. Але для цього спочатку необхідно формально описати мету поводження системи.

При описі поводження системи одним з найбільш загальних підходів [26] є звертання до екстремів принципам, коли мета поводження задається у виді прагнення до максимуму чи мінімуму деякої цільової функції (функціонала потенціалу):

$$V(x) = max\ .$$

Часто в ролі такого потенціалу виступає квадратична функція від змінних, що характеризують систему ("енергія" Хопфілда, сумарний квадрат помилки в методі зворотного поширення і т.п.). Вид цільової функції поряд з конфигурацією мережі є найважливішим фактором, що визначає характер поводження системи.

Визначимо узагальнену силу F_x діючу на змінну x і відповідальну за її зміни, як відособлена похідна від цільової функції по цій змінні:

$$F_x = dV / dX \tag{7.3}$$

Поняття сили зручно, тому що звичайно воно визначається таким чином, що має властивість аддитивности. Нехай на одну змінну діє кілька факторів (наприклад, вона зв'язана з декількома іншими змінними, як у нейроних мережах). Якщо необхідно визначити результат їхньої спільної дії, то слід охарактеризувати ці фактори через сили і потім знайти рівнодіючу цих сил; внаслідок аддитивности вона буде дорівнювати їх векторній сумі.

У загальному випадку картина поводження виглядає наступним чином: задаються цільова функція й умови, накладені на змінні. Зокрема, деякі змінні бути фіксованими. Тоді інші вільні змінні починають мінятися убік збільшення цільової функції. Процес продовжується доти, поки не буде досягнутий можливий у цих умовах максимум цільової функції (умовний максимум). Оскільки поводження вільної змінної завжди спрямоване на збільшення цільової функції, то для нього справедливо наступне вираження:

$$\frac{dV}{dt} = \frac{dV}{dx}\frac{dx}{dt} = F_x dx / dt > 0$$

Звідси випливає, що знак узагальненої сили F_x і знак реакції *dxldt* у вільному поводженні завжди збігаються: або обоє позитивні, або обоє негативні - тільки так можна забезпечити позитивність *dVldt.*

Таким чином, основне рівняння динаміки системи, що зв'язує швидкість зміни вільної змінної *dxldt* з діючої на неї силою F_x,, можна записати в наступному виді:

$$\text{sgn}(dx/dt) - \text{sgn}(F_x) = \text{sgn}(dV/dx) \tag{6.4}$$

Це рівняння є надзвичайно загальним. Усі відомі рівняння, що описують процеси навчання чи поводження нейронних мереж, є його окремими випадками і розрізняються між собою або характером змінних, або видом цільової функції.

В окремому випадку, коли змінні є безперервними і залежність між ними лінійна, рівняння може бути переписане в наступному виді:

$$Tdx/dt = F_x \tag{6.5}$$

Тут коефіцієнт пропорційності T має значення постійної часу, що характеризує інерційність змінної. Практично в нейроних мережах її задають довільно, керуючись бажанням забезпечити, з одного боку, досить високу швидкість, з іншого - стійкість процесу. У дискретному випадку завдання T визначає величину кроку процесу Δx, тобто зміна змінної за один такт. Чим більше T, тим менше крок і тим повільніше протікає процес. Основне рівняння динаміки в цьому випадку можна записати у виді:

$$T\Delta x = dV/dx$$

У випадку з двоїчними змінними цей крок визначений характером змінної: він або дорівнює 1 (якщо змінна приймає значення 0 і 1), або 2 (якщо її значення +1 і -1). У противному випадку рівняння динаміки може бути записане у виді:

$$x = \text{sgn}(dV / dx)$$

Усі типи рівнянь динаміки і різні їх комбанации використовуються в описі нейронних мереж.

Використовуємо як приклад мережу Хопфилда [26].

Цільова функція може бути описана у виді:

$$V = -E = -\sum_{ij} c_{ij} x_i x_j = \max$$

Змінні звичайно є двоїчними, вагарні коефіцієнти c_{ij} можуть розглядатися як дискретні чи безперевні змінні. Припустимо, що значення всіх змінних x_i фіксовані, вільними змінними є тільки ваги міжнейронних зв'язків c_{ij}. Зміна цих змінних означає навчання мережі. Основне рівняння динаміки в цьому випадку виглядає так:

$$T_c dc_{ij} / dt = F_c = dV / dc_{ij} = x_i x_j$$

Якщо c_{ij} - дискретні змінні, то рівняння прийме вид:

$$T_c \Delta c_{ij} = F_c = dV / dc_{ij} = x_i x_j$$

Це є не що інше, як правило Хебба для формування вагових коефіцієнтів у процесі навчання (з точністю до константи T_c).

Припустимо тепер, що фіксовано значення вагових коефіцієнтів c_{ij} (мережа навчена) і деякі з ознак x_j (задана часткова інформація про еталон). Для вільних ознак, що залишилися, основне рівняння динаміки приймає вид:

$$T_x dx_j / dt = F_j = dV / dc_{ij} = \sum_i c_{ij} x_i$$

Тут $F_j = \sum_i c_{ij} x_i$ і є рівнодіюча сил виду: $F_j = \sum_i c_{ij} x_i$, що діють на змінну x_j з боку змінних x_i, через зв'язки c_{ij}.

Якщо x_i- двоїчні змінні, що приймають значення +1,-1, то формула перетворюється для них у вираження:

$$x_j = \text{sgn} \sum_i c_{ij} x_i$$

У такому виді це рівняння динаміки спочатку і було написано Хопфілдом.

7.7. Навчання багатошарових мереж

Переваги багатошарових мереж були зрозумілі досить рано. Ясно було також, що для використання цих переваг, перетворення при переході від одного шару до іншого повинно бути нелінійним: послідовність лінійних перетворень дає знову лінійне перетворення з усіма його недоліками. Однак розвитку багатошарових мереж перешкоджало те, що не було теоретично обґрунтованого алгоритму навчання таких мереж. Неясно було, за яким правилом варто модифікувати зв'язки нейронів проміжних шарів, щоб одержати на виході потрібний результат.

Такий алгоритм був запропонований у ряді робіт [19, 20]. Зараз він відомий як метод "зворотного поширення помилки". Цей метод частковий случай основного рівняння динаміки.

Сутність його в наступному. Якщо відома цільова функція системи *V(x),* то можна знайти сили, що діють на будь-які змінні системи і породжують їхні зміни убік максимізації цільової функції. Якщо ці змінні безпосередньо входять до цільову функції, то ці сили обчислюються по формулі (6.3). Такі, наприклад, вихідні змінні системи, коли цільова функція безпосередньо залежить тільки від них.

Якщо ж цікавлячі змінні x_i- безпосередньо не входять до цільову функції, але зв'язані з змінними x_j-, що входять у неї, то діючі на x_i сили можуть бути обчислені за допомогою "закону передачі сили" - узагальнення відомого закону важеля на немеханічні системи. Закон, як відомо, говорить: що виграється в силі (*F*), те програється у відстані Δx. Це можна записати так:

$$F_i \Delta x_i = F_j \Delta x_j \; F, \text{ чи} \quad F_i / F_j = \Delta x_i / \Delta x_j = k_{ij}$$

Тут $\Delta x_i \;\; \Delta x_j$ - зміни двох зв'язаних змінних; F_i, F_J - діючі на них сили; k_{ij} - передатний коефіцієнт від x_i *до* x_j..

У нейронних мережах до числа змінних, що не входять безпосередньо в цільову функцію, відносяться параметри нейронів проміжних ("схованих") шарів. Однак змінні *i-го* рівня зв'язані з змінними наступного *j-го* рівня. Передатний коефіцієнт k_{ij} визначається насамперед вагою синаптичного зв'язку c_{ij}. Тому що якщо зв'язок нелінійний, то в нього повинна входити співмножником похідна від активаційної (передатної) функції, що залежить від значення вихідної змінної. Відповідно до закону передачі сили справедливо наступне співвідношення між силами, що діють на вихідні величини від y_i і y_j двох сусідніх шарів:

$$F_i = F_j k_{ij}$$

Підставляючи сюди вираження для сили і для передатного коефіцієнта та враховуючи, що змінна одного шару зв'язана з декількома змінними іншого, а діюча на неї сила повинна бути рівнодіючої суми сил, одержуємо наступне рекурентне співвідношення, що дозволяє знаходити похідну *n-1-го* шару по похідній *n-го* шару:

$$dV / dy_i^{(n-1)} = \sum\nolimits_j dV / dy_i g^{(n)} y_j^{(n)} (1 - y_j^{(n)}) c_{ij}$$

Знаючи силу, що діє на змінну, і використовуючи основне рівняння динаміки (6.4), можна написати закон зміни цієї змінної. Якщо змінна – це один з вагових коефіцієнтів c_{ij}- , то це і буде закон навчання.

У багатошарових мережах, як правило, вплив поширюється тільки в одному напрямку - від *i-го* шаруючи до *j-го*. Отже, матриця зв'язків несиметрична: тільки ваги c_{ij}-, можуть бути відмінні від нуля, тоді як c_{ji}, задані рівними нулю.

Розглянемо як приклад один з відомих алгоритмів навчання - "δ-правило". У найпростішому виді воно використовувалося вже при навчанні персептрона.

Суть його в наступному. Нехай у якості бажаного ("ідеального") значення вихідної величини задане значення y_j . Дійсне значення y^*_j виходить шляхом перетворення вхідних значень попереднього шару x і не обов'язково збігається з бажаним:

$$y_i = f(s_j) = f\sum_i c_{ij}x_i$$

Тут s_j - активаційна функція; $s_j = \sum_i c_{ij}x_i$.

Мета поводження складається в мінімізації квадрата помилки:

$$V = -E = -\sum_j (y_j - y^*_j)^2 / 2 = -\sum_j [y(\sum_i c_{ij}x_i) - y^*_j]^2 / 2 = \max$$

Тоді сила, що діє на змінну c_{ji} дорівнює:

$$F_c = -dE / dc_{ij} = -dE / dy_j dy_j / ds_j \qquad ds_j / dc_{ij} = \delta_j df / ds \cdot x_i$$

Тут через $\delta_j = (y^*_j - y_j)$ позначена різниця між ідеальним (бажаним) і дійсним значеннями вихідної величини. Звідси і назва алгоритму-
δ- правило. Швидкість модифікації ваги визначається основним рівнянням динаміки у формі (6.5):

$$T_c dc_{ij} / dt = F_c = \delta_j dy_j / ds_{jj} x_i$$

Це і є δ-правило. Якщо $F(s)$- функція сигмоідального виду, то виходить один з варіантів алгоритму зворотного поширення помилки:

$$T_c dc_{ij} / dt = \delta_j y_j (1 - y_j) x_i = (y_j - y^*_j) y_j (1 - y_j) x_i \qquad (6.6)$$

Відповідно до цього алгоритму швидкість модифікації вагового коефіцієнта c_{ij} пропорційна трьом факторам:

- "помилці " - різниці між дійсним і бажаним значеннями вихідної величини $(y^*_j - y_j)$;

- похідної від функції активації $y_j(1-y_j)$;
- вхідній величині x_t.

Якщо залежність від першого фактора є корисною, то два останніх фактори служать джерелом різного роду неприємностей, що виникають у процесі навчання.

В даний час за звичай використовують різні модифікації цього алгоритму, мета яких поліпшити стійкість процесу навчання і дозволити ряд інших проблем. Наприклад, вводять в алгоритм "пам'ять " про попередній крок, що додає йому певну інерційність і стійкість до перешкод.

7.8. Проблеми і перспективи

Зупинимося на труднощах, зв'язаних з навчанням нелінійних нейронних мереж. Основні з них наступні [57].

Повільна збіжність процесу навчання. Строга збіжність доведена для диференціальних рівнянь, тобто для нескінченно малих кроків у просторі ваг. Але нескінченно малі кроки означають нескінченно великий час навчання. При кінцевих кроках збіжність не гарантується, але навіть якщо вона має місце, то потрібний для цього час може бути занадто великим, порівнянним з часом життя користувача.

"Пастки", що створюютьсяі локальними мінімумами. Детермінований алгоритм навчання не в силах знайти глобальний мінімум чи залишити локальний мінімум. Одним із прийомів, що дозволяють обходити пастки, є розширення розмірності простору ваг за рахунок збільшення числа нейронів другого шару. Деякі нові можливості відкривають стохастичні методи. Але все це досягається ціною додаткових витрат часу навчання.

"Параліч" мережі. Сигмоідальний характер передатної функції нейрона приводить до того, що якщо в процесі навчання кілька ваг стали занадто великими, то нейрон попадає на горизонтальну ділянку функції в область насичення. При цьому, зміни інших ваг, навіть досить великіх, практично не позначаються на величині виходу нейрона, а виходить, і на величині цільової функції.

З вираження для похідної від передатної функції (6.2) видно, що вона прагне до нуля, коли у наближається до нуля чи одиниці. Це значить, що зв'язок між сусідніми шарами практично розривається, і процес навчання блокується.

Невдалий вибір діапазону вхідних змінних - досить елементарна, але часто чинена помилка. Якщо x_i - двоїчна змінна зі значеннями 0 і 1, то приблизно в половині випадків вона буде мати нульове значення: $x_i = 0$. Оскільки x входить співмножником у вираження для модифікації ваги (6.6), ефект буде такий саме, що при насиченні: модифікація відповідних ваг припиниться, і навчання буде блоковано. Правильний діапазон для вхідних змінних повинен бути симетричним, наприклад від + 1 до -1.

*"Перетренування", в*арто мати на увазі, що зайво висока точність, отримана на навчальній вибірці, може обернутися нестійкістю результатів на тестовій вибірці. Тут діє загальний закон: чим краще система адаптована до даних конкретних умов, тим менше вона здатна до узагальнення й екстраполяції, тим скоріше вона може виявитися непрацездатною при зміні цих умов. А такі зміни від вибірки до вибірки неминучі, особливо якщо вибірки невеликого розміру. Розширення обсягу навчальної вибірки дозволяє домогтися більшої стійкості, але за рахунок збільшення часу навчання.

Проблема обсягу пам'яті. Ємність пам'яті нейронних мереж, її здатність зберігати і відтворювати інформацію є однією з найважливіших характеристик. Однак якщо в традиційних-послідовних машинах характеристики пам'яті досить зрозумілі і доступні оцінці, то в нейронних мережах справа обстоїть набагато складніше.

Стохастичні методи навчання. Детерміністичний метод навчання робить модифікацію ваг мережі тільки на основі інформації про напрямок градієнта цільової функції в просторі ваг. Такий метод здатний привести до локального екстремуму, але не здатний вивести з нього, оскільки в точці екстремума сила обертається в нуль і причина руху зникає (як це видно з рівняння динаміки (6.4). Щоб змусити мережу залишити локальний екстремум і відправитися на пошуки глобального, потрібно створити додаткову силу, що залежала б не від градієнта цільової функції, а від якихось інших факторів. Вибір цих факторів, більш-менш виправданий різними

евристичними розуміннями, і складає основу різних методів подолання локальних пасток. Один з найпростіших методів полягає в тому, щоб просто створити випадкову силу і додати її до детерміністичної. Сама присутність такого роду випадкових факторів: "шуму", "температури" приводить до "усереднення, згладжуванню, розмивання" потенційних бар'єрів. Дрібні гребені і западини зникають, і якщо в просторі параметрів є глобальний екстремум, виявляється сила, що діє в напрямку цього екстремума. Правда, сила ця має випадковий характер: вона тільки в середньому спрямована убік цього екстремума. В міру наближення до нього ця середня регулярна складова зменшується, наближаючись до нуля, і залишається тільки випадкова. Навіть досягши глобального екстремума, система буде продовжувати коливатися біля нього з досить великою амплітудою. Тому за звичай, в міру наближення до экстремуму, амплітуду випадкової складовий поступово знижують. Така процедура нагадує отжиг металу, коли для досягнення оптимальної енергетичної структури металу його спочатку нагрівають, а потім повільно і поступово охолоджують. Цей метод одержав назву "метод імітації отжигу ".

7.9. Застосування нейромережних технології

В даний час відомо багато вдалих прикладів застосування нейромережного підходу [28] для побудови інтелектуальних інформаційних і, зокрема, експертних систем.

Комбіноване використання експертної системи й апарата штучних нейронних мереж забезпечує необхідну гнучкість і самонавчання на основі знань, у той же час, отримані від експертів знання дозволяють істотно спростити структуру нейронних мереж, зменшити число нейронів і зв'язків у мережі.

Наприклад, медичні нейромережні експертні системи виявили себе як серйозний суперник традиційних експертних систем, що складають конкуренцію кваліфікованим експертам. Дослідження, проведені з використанням розробленої

нейромережної експертної системи, показали її чудові можливості по діагностиці деяких класів хвороб, що погано діагностуються лікарями.

Результати перевірки свідчать про високу вірогідність результатів, що досягаються такими системами- до 94%.

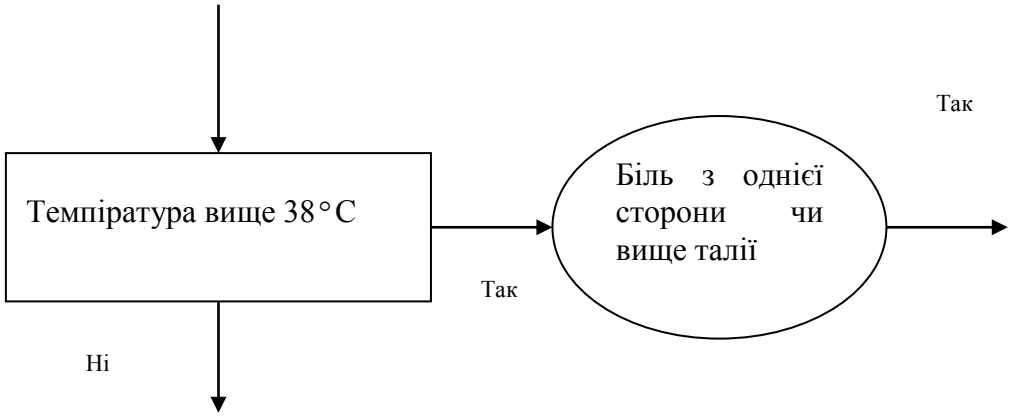
Розглянемо приклад постановки задачі для експертної системи. Багатьох людей турбують болі в спині, що часто виникають раптово і без визначеної причини. Звичайно важко визначати характер нездужання через відсутність ознак будь-якого конкретного захворювання. Біль звичайно зникає після короткого відпочинку. Лікарі називають це невизначеними болями в спині. Такі нездужання - основна причина загублених робочих днів. Люди, що займаються важкою фізичною працею, зв'язаною, наприклад, з підняттям ваги, більш піддані періодично виникаючим болям у спині. Це може також турбувати тих, хто проводить багато часу без руху. У будь-якому випадку, болі в спині можуть мати багато причин. Велике значення при цьому має правильно поставлений діагноз.

Метою побудови експертної системи є діагностика захворювання, що може бути причиною періодичних болів у спині.

Принцип побудови експертної системи на базі нейронної мережі полягає в наступному. Складаються питання, відповіді на який мають бінарний вид, тобто «Так» чи «Ні». При складанні «вектора опитування», якщо при діагностиці випливає відповідь «Так», те компоненту вектора привласнюється 1, якщо «Ні», 0.

Відповідно до вищесказаного, вирішальне «дерево», приведене на мал. 7.5, може бути записане у виді трьох векторів:

{..., 1,1, 0,...},{,..,1,0,1,...} і {..., 0,...}.



Малюнок 7.5 Частина вирішального дерева

Перші два записи (вектори) передають наступний зміст:

ЯКЩО:

Пацієнт має температуру вище 38° С, і почуває біль тільки з однієї сторони спини

ЧИ:

Пацієнт почуває нездужання.

Третій запис (вектор) передає:

ЯКЩО:

Пацієнт має температуру менше 38°С.

ТО: ....

Складемо прості питання для діагностики можливих причин болі у спині. Вхідні вектори в лінгвістичній формі будуть при цьому мати наступний вид:

1.1) Біль виникає після підйому ваги?

І/ЧИ:

1.2) Після фізичної вправи, що виснажує?

2) Температура вище 38 ºС?

3.1) Пацієнт старше 60 років?

І/ЧИ:

3.2) Пацієнт провів кілька тижнів у ліжку чи в інвалідному кріслі?

4) Пацієнт старше 45 років?

5) Біль сильніше ранком?

6.1) Пацієнту заважає біль при ходьбі?

ЧИ:

6.2) Біль почувається тільки в одній нозі?

7) Біль обмежений головним чином у спині?

8.1) Біль - тільки з однієї сторони спини, вище талії?

8.2) Почуття нудоти?

9) Біль набагато сильніше з однієї сторони хребта?

10) Звичайно хворіє шия чи спина між плічми?

Аналогічно можна побудувати вектор вихідних значень, керуючись тим же самим правилом, що і для вхідних. Якщо на виході нейронної мережі, що відповідає якому-небудь діагнозу одержуємо 1, то на даний діагноз варто звернути увагу, тому що він може бути причиною болю.

Вектор вихідних змінних (діагноз) має наступні компоненти:

1) Ішіас, викликаний тиском на корінь сідничного нерва; необхідна консультація лікаря.

2) Можливий люмбаго (простріл), ймовірно викликаний сильною напругою спини.

3) Можлива інфекція нирок, чи болі можуть бути супроводом загального вірусного запалення; необхідна термінова консультація лікаря.

4) Біль у спині (можливо дуже сильна), може бути наслідком будь-якого вірусного захворювання, наприклад грипу; необхідна консультація лікаря.

5) Можливе ушкодження кісти в результаті травми; необхідна консультація лікаря.

6) Можливий артрит хребців шиї.

7) Можливий остеоартрит у нижній частині грудей, нирок чи хребта.

8) Можливо хронічне запалення суглобів.

9) Причина болю не з'ясована; необхідна консультація лікаря.

Повне вирішальне «дерево» для експертної системи приведене на мал. 6.2. Мережа була реалізована за допомогою нейропакета NeuraPlanner. Навчання відбувалося за допомогою 18 навчальних векторів. Для опитування мережі необхідно закодувати в двоїчному виді питання, а потім проробити зворотну операцію з відповідями. Час навчання мережі при заданій помилці 0,05 склав близько 2 хв. Мережа навчилася за 7500 циклів.

Для перевірки отриманих результатів було проведене наступне опитування мережі.

Постановка і кодування питань:

1.1) Біль виникає після підйому ваги? Так – 1.

І/ЧИ:

1.2) Після фізичної вправи, що виснажує? Так -1.

2) Температура вище 38°С? Ні - 0.

3.1) Пацієнт старше 60 років? Ні - 0

І/ЧИ:

3.2) Пацієнт провів кілька тижнів у ліжку чи в інвалідному кріслі?

Ні - 0.

4)Пацієнт старше 45 років? Ні - 0.

5) Біль сильніше ранком? Так - 1.

6.1) Пацієнту заважає біль при ходьбі? Так - 1

ЧИ:

6.2) Біль почувається тільки в одній нозі? Ні - 0.

7) Біль обмежений головним чином у спині і не поширюється де-небудь ще? Так -1.

8.1) Біль - тільки з однієї сторони спини, вище талії? Так- 1.

И:

8.2) Почуття нудоти? Ні - 0.

9) Біль набагато сильніше з однієї сторони хребта? Ні-0.

10) Звичайно хворіє шия чи спина між плічми? Ні-0.

Ідеї в області нейромережної технології швидко знайшли своє застосування в новому типі мікроелектронної техніки– нейропроцесорах і нейрокомп`ютерах (НК).

Зараз нейросмережну технологію застосовують у багатьох областях промисловості.

Читання друкованого тексту. Оптична система розпізнавання (Optical Character Recognition) фірми Sharp Corp. використовується для розпізнавання японських ієрогліфів, містить порядку 10 млн. зв'язків і використовує різновид системи LVQ Кохонена. Система Onyx Check Reader фірми VeriFon Inc. забезпечує

точне і недороге зчитування чисел на чеках, використовуючи стандартний аналоговий нейрочіп фірми Synaptics.

Фірма NEC (Японія) оголосила, що створила пристрій для візуального розпізнавання букв. Точність розпізнавання перевищила 99%. Успіх був досягнутий за рахунок інтеграції звичайних алгоритмів з нейромережею, що працює за методом зворотного поширення помилки.

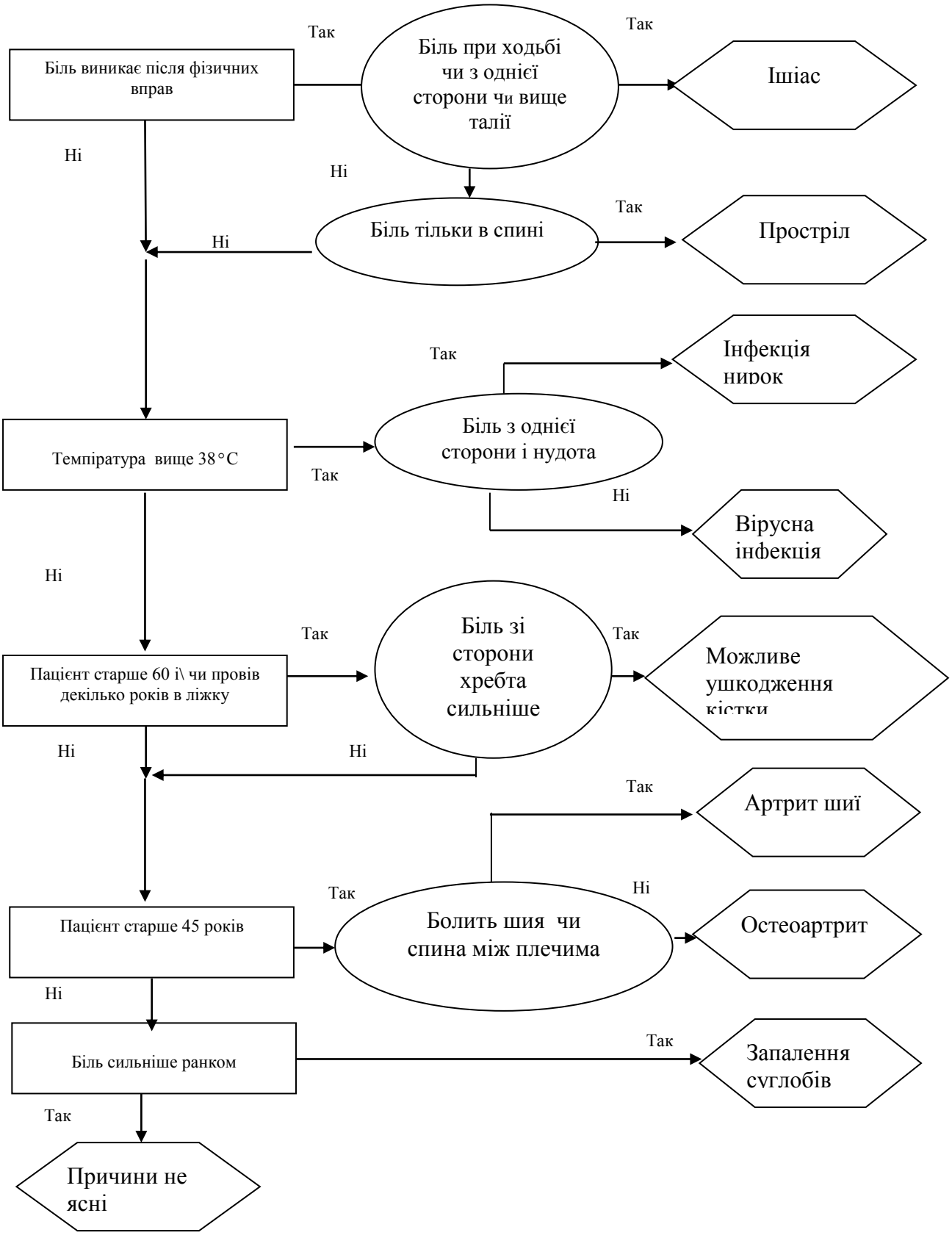
Фірма Poget Computer використовує мережу NestorWriter для розпізнавання рукописних символів на персональних комп'ютерах з пір'яним введенням. Система Automated Data Entry System (Hecht – Nielsen Corp., США) була використана для обробки чеків.

В університеті Дж.Гопкінса (США) створена нейронна мережа "Net-Talk", призначена для читання вголос друкованого тексту (300 нейронів, 10 000 зв'язків, слова створюються синтезаторами). За вісім днів мережа освоїла 20 000 англійських слів. За свідченням очевидців, звучання тексту дуже нагадує голос дитини на різних етапах навчання мови.

Аналіз спектрографічних даних у хімічній промисловості, класифікація дефектів гучномовців (CTS Electronics), оцінка чистоти апельсинового соку (Florida Departament of Citrus).

Фірма Neuromedical System Inc. пропонує електроенцефалографи, апаратуру для скрінінгу рака й інше устаткування, засноване на нейромережній технології.

Фірма Promised Land Brothers INC. пропонує недорогий пакет по оцінці ефективності інвестицій. Chase Manhatten Bank використовує гібридну систему розпізнавання образів з нейромережею для оцінки ризику при видачі позик.



Малюнок 7.6 Повне вирішальне дерево

LABELS	1.1	1.2	2	3,1	3,2	4	5	6,1	6,2	7	8,1	8,2	9	10	1	2	3	4	5	6	7	8	9
TESTING	>	>	>	>	>	>	>	>	>	>	>	>	>	>	<	<	<	<	<	<	<	<	<
dummy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
END																							
INTERROGATING	>	>	>	>	>	>	>	>	>	>	>	>	>	>	<	<	<	<	<	<	<	<	<
dummy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	1	1	0	1	1	0	0	0	?	?	?	?	?	?	?	?	?
END																							
TRAINING	>	>	>	>	>	>	>	>	>	>	>	>	>	>	<	<	<	<	<	<	<	<	<
dummy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
4	1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
7	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
8	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
10	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0
11	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
12	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
13	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
14	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
15	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
16	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
17	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
END																							
LIMITS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Higth	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Lows	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
END																							

Малюнок 7.7 Файл з навчальною вибіркою

Інтелектуальний контролер на основі нейромережевої технології для керування дуговою піччю, установлений фірмою Neural Application Corp., дозволяє за один рік зберегти кошти на придбання ще однієї печі. Техаська фірма Ruget Sound Refinery включила нейромережі в систему керування очищенням нафти.

Фірма US Naval Air Warfare Center (США) використовує нейромережі для керування снарядами. Там, де вимагаються швидкі рішення, нейромережі мають величезні переваги перед звичайними методами. Фірма Lockheed (США) розробила систему керування повітряним боєм для винищувача, засновану на прогнозуванні можливих дій супротивника. Система використовує нейромережу для інтеграції багатоканальних даних про зразки польоту і повітряного бою.

Контрольні питання

1. Припустимо,

8. Інструментальний комплекс G2

Історія розвитку інструментальних засобів (ІС) для створення ЕС реального часу почалася в 1985 р., коли фірма Lisp Machine Inc. випустила систему Picon для символьних ЕОМ Symbolics. Успіх цього ІС привів до того, що група ведучих розроблювачів Picon у 1986 р. утворила приватну фірму Gensym, що, значно розвивши ідеї, закладені в Picon, у 1988 р., вийшла на ринок з ІС під назвою G2, версія 1.0. Сьогодні їй належать 50% світового ринку експертних систем, що використовуються у системах керування.

З відставанням від Gensym на 2 - 3 роки інші фірми почали створювати свої ІС для ЕС РВ. З погляду незалежних експертів NASA, що проводили комплексне дослідження характеристик і можливостей деяких з перерахованих систем, у даний час найбільш просунутим ІС, безумовно, залишається G2. Наступні місця зі значним відставанням (реалізовано менш 50% можливостей G2) займають RTWorks- фірма Talarian (США), COMDALE/C (Comdale Techn. - Канада), COGSYS (SC - США), ILOG Rules (ILOG - Франція).

Класи задач, для яких призначена G2 [57] і подібні їй системи:

- моніторинг у реальному масштабі часу;
- системи керування верхнього рівня;
- системи виявлення несправностей;
- діагностика;
- складання розкладів;
- планування;
- оптимізація;
- системи - порадники оператора;
- системи проектування.

Інструментальні засоби фірми Gensym є еволюційним кроком у розвитку традиційних експертних систем від статичних предметних областей до динамічних.

Основним достоїнством оболонки експертних систем G2 для користувачів є можливість застосовувати її як інтегруючий компонент, що дозволяє за рахунок

відкритості інтерфейсів і підтримки широкого спектра обчислювальних платформ легко об'єднати вже існуючі, розрізнені засоби автоматизації в єдину комплексну систему керування, що охоплює всі аспекти виробничої діяльності - від формування портфеля замовлень до керування технологічним процесом і відвантаження готової продукції. Набір інструментальних середовищ фірми Gensym, згрупований по проблемній орієнтації, охоплює всі стадії виробничого процесу і виглядає так:

- інтелектуальне керування виробництвом- G2 Diagnostic Assistant (GDA), NeurOn-Line (NOL), Statistical Process Control (SPC), BatchDesign_Kit;
- оперативне планування- G2 Scheduling Toolkit (GST), Dynamic Scheduling Packadge (DSP);
- розробка і моделювання виробничих процесів- ReThink, BatchDesign_Kit;
- керування операціями і корпоративними мережами - Fault Expert.

G2 функціонує на більшості існуючих платформ (VMS, ULTRIX, Open VMS, OSF/1, Windows 95, 98, NT і т.д., Sun OS/Solaris 1, 2.x, HP-X, AIX, DG/UX, IRIX, UNIX, EWS-UX/V). База знань G2 зберігається в звичайному ASCII-файлі, що однозначно інтерпретується на кожній з підтримуваних платформ. Розглянемо основні компоненти G2- базу знань, машину виводу, систему моделювання і планувальника.

8.1. База знань

Усі знання в G2 зберігаються в двох типах файлів: бази знань (БЗ) і бібліотеки знань (Біз). У файлах БЗ зберігаються знання про додатки: визначення всіх об'єктів, об'єкти, правила, процедури і т.п. У файлах Біз зберігаються загальні знання, що можуть бути використані більш ніж в одному додатку, наприклад визначення стандартних об'єктів. Файли БЗ мають розширення kb (knowledge base), а файли Біз - kl (knowledge libraries). Файли БЗ можуть шляхом заміни розширення перетворитися в Біз і назад.

З метою забезпечення повторного використання додатків у G2 реалізований засіб, що дозволяє поєднувати раніше створені kb-и kl-файли з поточним додатком.

При цьому G2 автоматично виявляє і виводить на дисплей конфлікти в поєднуваних знаннях.

Знання в G2 структуються такими способами: ієрархія класів, ієрархія модулів, ієрархія робочих просторів. Клас є основою представлення знань у G2. Поняття "клас у G2" базується на об`єктно-орієнтованої технології (ООТ).

В ООТ структури даних представляються у виді класів об'єктів (визначень об'єктів), що мають визначені атрибути. Класи успадковують атрибути від суперкласів і передають свої атрибути підкласам. Кожен клас (крім кореневого) може мати конкретні екземпляри класу. У четвертій версії G2 введен механізм множинного спадкування. Тепер у системі досить легко зробити, наприклад, новий клас саморегулюючих насосів від класів контролерів і насосів. У системі вирішена проблема конфліктів між іменами атрибутів. Використання ООТ забезпечує наступні переваги:

1) зменшує надмірність і спрощує визначення класів, тому що визначається не весь клас, а тільки його відмінності від суперкласу;

2) дозволяє використовувати загальні правила, процедури, формули, що зменшує їхню кількість;

3) є природним для людини за способом опису сутностей.

8.2. Сутності й ієрархії класів

Клас у G2 є основою представлення знань. Усе, що зберігається в БЗ і чим оперує система, є екземпляром того чи іншого класу. Усі синтаксичні конструкції G2 теж є класами. Для збереження спільності навіть базові типи даних- символьні, числові, булеві і дійсні значення нечіткої логіки представлені відповідними класами. Опис класу (теж екземпляр спеціального класу) включає посилання на суперклас (is-a-ієрархія) і перелік атрибутів, специфічних для класу (part-of-ієрархія).

Концептуально ієрархія класів G2 берет свій початок від кореневого класу, іменованого item-or-value (сутність чи значення). Клас item-or-value сам по собі не може мати екземплярів. Однак тому що він є коренем всієї ієрархії класів, він визначає основне поводження всіх класів G2. Item-or-value має дві похідні класу - value (хоча концептуально галузь value представляється класом, у дійсності це типи даних G2) і item. Кожний з цих класів має свої похідні класи. Сутність (item) є коренем розгалуженої ієрархії класів. Найбільш важливі галузі цієї ієрархії можуть бути згруповані в невелике число категорій.

Ієрархія модулів і робочих просторів

G2-додаток не являє собою єдиний блок. Він структується за допомогою модулів і робочих просторів на легко керовані шматки. Незважаючи на те, що функції модулів і робочих просторів схожі, між ними є істотні розходження.

Додаток у G2 може бути організован у виді однієї чи декількох БЗ, що називаются модулями. В останньому випадку говорять, що додаток модуляризовано (структурований на модулі). Модулі додатка організовані в деревоподібну ієрархію з одним модулем верхнього рівня. Модулі наступного рівня складаються з тих модулів, без яких не може працювати модуль попереднього рівня. Ці модулі називають "безпосередньо необхідні модулі".

Існують 2 способи створити G2-додатка.

1. Розробляється одномодульний додаток, що потім при необхідності розділяється на окремі модулі.

2. Додаток з самого початку створення складається з декількох модулів. Деякі з цих модулів розробляються вперше, а інші можуть вибиратися з бібліотеки знань.

Структурування додатка на модулі забезпечує наступні переваги:

- дозволяє розробляти додаток одночасний декільком групам розроблювачів;
- спрощує розробку, налагодження і тестування;
- дозволяє змінювати модулі незалежно друг від друга;
- спрощує повторне використання знань.

Робочі простори є контейнерним класом, у якому розміщаються інші класи і їхні екземпляри, наприклад об'єкти, зв'язки, правила, процедури і т.д. Кожен модуль (база знань) може містити будь-яку кількість робочих просторів. Робочі простори утворюють одну чи кілька деревоподібних ієрархій з відношенням is-a-part-of (є частиною). З кожним модулем (базою знань) асоціюється одне чи кілька робочих просторів верхнього (нульового) рівня, кожне з цих робочих просторів є коренем відповідної деревоподібної ієрархії. У свою чергу, з кожним об'єктом (визначенням об'єкта чи зв'язку), розташованим у нульовому рівні, може бути асоційований робочий простір першого рівня, зв'язаний з ним відношенням "є частиною", і т.д.

Розходження між модулями і робочими просторами полягає в наступному. Модулі розділяють додаток на окремі бази знань, спільно використовувані в різних додатках. Динамічні модулі (аналог бібліотек динамічного зв'язування) можуть довантажуватися і витіснятися з оперативної пам'яті під час виконання програмно й одночасно використовуватися декількома додатками. Робочі простори виконують свою роль при виконанні додатка. Вони містять у собі (і у своїх підпросторах) різні сутності і забезпечують розбивку додатка на невеликі частини, що легше зрозуміти й обробляти. Наприклад, весь процес розбивається на підпроцеси, і з кожним підпроцесом асоціюється свій підпростір.

Робочі простори можуть установлюватися (вручну чи дією в правилі-процедурі) в активний чи неактивний стан (тобто сутності, що знаходяться в цьому просторі й у його підпросторах, стають невидимими для механізму виводу). Механізм активації (деактивації) робрчих просторів використовується, наприклад, при наявності альтернативних груп правил, коли активною повинна бути тільки одна з альтернативних груп.

Крім того, робочі простори використовуються для завдання користувальницьких обмежень, що визначають поводження додатка для різних категорій користувачів.

8.3. Структура даних БЗ

Глобально сутності в БЗ G2 з погляду їхнього використання можуть бути розділені на структури даних і твердження, що виконуються. Прикладами перших є об'єкти і їхні класи, зв'язки (connection), відносини (relation), змінні, параметри, списки, масиви, робочі простори і т.п. Прикладами других - правила, процедури, формули, функції і т.п.

Опишемо найбільш важливі галузі ієрархії "item".

Об'єкт (object) і його підкласи. Об'єкти представляють об'єкти реального світу в додатку. Клас об'єктів визначає атрибути, що дозволяють створити піктограми для об'єктів, визначити їхнє положення на схемах і *відростки зв'язків (stubs)* для приєднання їх до інших об'єктів.

Зв'язок (connection). Клас для зображення шляхів між об'єктами. Можна створити підклас зв'язків для зазначення різних типів потоків, що можуть існувати між об'єктами на схемі. Наприклад, об'єкти можуть бути з'єднані водопровідними трубами і (чи) проводами, що передають логічні сигнали. Визначивши різні класи для цих зв'язків, можна бути упевненим, що G2 буде їх розрізняти і ніколи не дозволить воді текти по електропроводах.

Робочий простір БЗ (Kb-workspace). Клас, що визначає незалежний сегмент бази знань, який може бути активований чи деактивирован. Робочі простори відображаються як окремі, обмежені робочі області, у яких можна поміщати об'єкти і поєднувати їх у схеми. Можна створити зв'язок між робочими просторами за допомогою точок *зв'язку (connection posts).* По суті клас робочих просторів є розвитком концепції робочої пам'яті в традиційних системах. Можна сказати, що робоча пам'ять системи G2 будується на основі ієрархії робочих просторів. Ієрархія робочих просторів тісно зв'язана з графічним представленням об'єктів. Робочий простір є контейнерним класом для екземплярів інших класів. Кожен екземпляр об'єкта може володіти своїм робочим простором, що представляє його внутрішню структуру.

Введення концепції робочих просторів забезпечує дві важливі функції системи G2: можливість здійснювати міркування на різних рівнях абстракції і можливість тривалої (теоретично - нескінченної) роботи системи без необхідності "зборки сміття" у межах

відведеного обсягу оперативної пам'яті, що дуже важливо для систем керування безупинними процесами.

Класи користувальницького інтерфейсу (user-interface). Визначають такі елементи користувальницького інтерфейсу, як меню, селективні кнопки (radio button), повідомлення (message), шкали, кругові шкали і багато чого іншого. Можна визначати підкласи класу повідомлень, наприклад, для створення повідомлень зі спеціальним способом відображення. З усіх класів користувальницького інтерфейсу тільки для повідомлень існує можливість створювати похідні класи.

Класи описів класів (class definition) визначають класи, екземпляри яких містять створені користувачем описи класів і служать шаблонами для створення екземплярів інших класів. Ці класи породжені від *класу опису (definition).* Опис має три підкласи- *опис об'єкта (object-definition), опис зв'язку (connection-definition)* і *опис повідомлення (message-definition)* відповідно до класів, які може визначати користувач.

Класи мови G2 (G2 language): ці класи використовуються для визначення різних елементів мови G2, таких, як правила, відносини, дії і процедури. Не можна створити власні похідні класи від цих класів.

За допомогою G2 нові класи можуть створюватися не тільки в процесі розробки, але і динамічно, під час роботи додатка. Під час виконання додатка може бути створений, модифікований чи знищений екземпляр будь-якого класу чи цілий клас. Це стосується як об'єктів, так правил і процедур. У цьому розумінні G2 більш об'єктно-орієнтована система, чим навіть C++. Ця можливість є частиною загальних можливостей G2, що доповнює описи класів і що дозволяють створювати нові сутності, включаючи робочі простори, правила, зв'язки і процедури. G2 забезпечує операції *create by cloning (створення клонуванням)* і *change the text of (змінити текст),* що використовуються для клонування схожого опису класу і наступного його редагування відповідно до вимог нових особливостей. За замовчуванням динамічно створені сутності є *тимчасовими (transient),* тобто вони існують тільки протягом даного сеансу роботи і не зберігаються в базі знань. Однак опис класу повинен бути *постійною (permanent)* сутністю в момент створення екземпляра чи похідного класу. Можна використовувати операцію *make permanent (зробити постійним)* для перетворення тимчасової сутності опису класу в постійну.

Усі класи G2 володіють принаймні однією загальною властивістю - їхні екземпляри мають графічну форму представлення. Використовуючи ці графічні образи разом із класом зв'язків, можна будувати схеми систем для будь-якого рівня складності. Крім візуалізації взаємодії об'єктів G2 надає синтаксичні конструкції, що дозволяють здійснювати міркування на основі графічних схем. Наприклад, можна перевірити стан усіх вентилів, з'єднаних з даною ємністю, чи визначити температуру об'єкта, найближчого до зазначеного.

Розглянемо докладніше найбільш важливі класи сутностей.

Виділяють об'єкти (класи), вбудовані в систему і вводимі користувачем. При розробці додатка, як правило, створюються підкласи користувальницьких і вбудованих класів, що відбивають специфіку даного додатку. Серед вбудованих підкласів найбільший інтерес представляє підклас об'єктів, що включає підкласи змінних і параметрів, і підклас зв'язків (connection) і відносин (relation).

8.4. Об'єкти

Об'єкти в базі знань являють собою відображення елементів реального світу, що будуть застосовуватися при рішенні поставленої перед ЕС РВ задачі. Виділяють постійні і тимчасові об'єкти. *Постійні об'єкти* заносяться в БЗ розроблювачем ЕС РВ у процесі діалогу із системою, у той час як *тимчасові об'єкти* створюються після виконання спеціальних команд у правилах і процедурах. Тимчасові об'єкти можуть існувати в БЗ тільки в процесі роботи ЕС РВ. З кожним об'єктом асоціюється таблиця атрибутів, у яку заносяться істотні для розв'язуваної задачі властивості об'єкта. Елемент даної таблиці являє собою пари "атрибут -значення".

Об'єкти можуть мати графічні образи, відображувані на екрані дисплея, що називаються *піктограмами.* На піктограмах розроблювачем можуть бути виділені окремі ділянки. Колір таких ділянок може змінюватися в результаті виконання спеціальних команд у правилах чи процедурах. Таким способом забезпечується висока наочність інформації, що надається користувачу.

Оскільки реальні додатки можуть містити велику кількість об'єктів, доцільно надавати можливість об'єднання об'єктів зі схожими властивостями в класи. Класи об'єктів складають ієрархію, у якій визначається відношення "батьківський клас - підклас". Об'єкти підкласів можуть успадковувати атрибути і піктограми батьківських класів.

Ієрархічна упорядкованість класів значно спрощує задачу визначення нових класів у додатку. Наприклад, атрибути, що характеризують об'єкти різних класів, можуть бути однократно визначені в одному класі, що є загальним батьківським класом для них. Такі атрибути будуть автоматично успадковуватися об'єктами, що належать до підкласів, знімаючи необхідність їхнього повторного визначення. Іншим важливим достоїнством введення класів-об'єктів є можливість складання правил, що відносяться до всіх об'єктів, які належать до деякого класу (загальних правил). Задача розроблювача значно спрощується за рахунок того, що їм може бути складений ряд загальних правил, застосовних до різних класів-об'єктів додатка, а результуюча БЗ буде мати менший обсяг у порівнянні з БЗ, у якій не можуть застосовуватися загальні правила.

Особлива роль у G2 приділяється змінним. На відміну від статичних систем змінні в G2 поділяються на три види: власне змінні, параметри і прості атрибути. *Параметри* - одержують значення в результаті роботи машини виводу чи виконання якої-небудь процедури. *Змінні* представляють вимірювані характеристики об'єктів реального світу і тому мають специфічні риси: час життя значення і джерело даних. *Час життя значення змінної* визначає проміжок часу, протягом якого це значення актуальне, після закінчення цього проміжку вважається, змінна не має значення. На відміну від змінних параметри завжди мають значення, тому що їхні значення або задані як початкові значення, або перераховані механізмом виводу G2.

Оскільки системі може знадобитися поточне значення змінної, для кожної з них повинно бути визначене *джерело даних (сервер даних).* Джерелом даних для змінної можуть служити: машина виводу, підсистема імітаційного моделювання чи зовнішнє стосовно G2 джерело даних. З змінними можуть бути асоційовані формули імітаційного моделювання, у результаті застосування яких система також

може одержувати значення змінних. Для параметрів зазначений механізм одержання значень із джерела даних не використовується; вони одержують нові значення після виконання спеціальних операторівв заключеннях правил чи процедур.

При посиланні в правилі чи процедурі як для змінних, так і для параметрів припустиме використання наступних виражень, що відбивають динаміку їхніх значень:

- поточне значення;
- значення в заданий момент часу;
- середнє значення за інтервал часу;
- інтеграл інтервалу часу;
- інтерполяція значення в заданий момент часу;
- максимальне (мінімальне) значення за інтервал часу;
- кількість зібраних значень за інтервал часу;
- швидкість зміни значень протягом інтервалу часу;
- стандартне відхилення протягом інтервалу часу.

Очевидно, що далеко не для усіх використовуваних у додатку значень потрібно застосовувати такий могутній інструментарій, тому з метою підвищення ефективності функціонування системи в цих випадках використовують прості атрибути.

8.5. Зв'язки і відносини

G2 передбачені два види взаємозв'язків між об'єктами: зв'язки і відносини. Під *зв'язками* розуміється взаємозв'язок між двома сутностями, що задається розроблювачем додатка і має графічне представлення. У реальному фізичному оточенні, описуваному в G2, зв'язку може відповідати фізичний зв'язок між сутностями, такий, як електричне з'єднання чи трубопровід. У G2 розроблювач може задавати класи зв'язків, посилатися на об'єкти за допомогою вказівки зв'язків, у яких вони беруть участь, а також робити вивід на підставі наявності чи відсутності зв'язків. Відносини, як і зв'язки, представляють взаємозв'язок між

об'єктами. Під *відношенням* розуміється пойменований взаємозв'язок між двома сутностями. G2 надає можливість розроблювачу задавати різні типи відносин. На підставі наявності чи відсутності відносини між об'єктами можуть відбуватися виводи. Основні відмінності зв'язків і відносин зводяться до наступного:

- зв'язки задаються розроблювачем у процесі створення ЕС, у той час як відносини встановлюються динамічно після виконання спеціальних операторів у правилах чи процедурах;
- зв'язки мають графічне представлення, у той час як відносини не відображаються на екрані дисплея. G2 графіка - це більше ніж зображення. G2 графіка може моделювати знання, що представляють об'єкти, зв'язки і залежності між об'єктами. G2 може міркувати в термінах зв'язку, дотримуючись мережі зв'язаних об'єктів для визначення причин і результатів. Графічна зв`зність об'єктів G2 дозволяє розширити прикладну програму використовуючи графічне об'єднання аналогів. Графіка включає вбудовані діаграми (графіки), таблиці і малюнки і т.д. G2 також працює з утилітами графічного інтерфейсу Windows. Ці утиліти використовують усі переваги объектно-ориентированных можливостей.
- відносини на відміну від зв'язків недоцільно зберігати як постійну частину БЗ.

8.6. Твердження БЗ, що виконуються

Основу тверджень БЗ, що виконуються, складають правила і процедури. Крім того, є формули, функції, дії і т.п. Правила в G2 мають традиційний вид: умова (антецедент) і висновок (консеквент). Крім if-правила: умова ("if <логічне вираження>") і висновок ("then <дії>") використовуються ще 4 типи правил: initially, unconditionally, when і whenever.

Способи застосування кожного правила визначаються його синтаксисом:

<правило> ::= *{\<префікс for>*
{<правило if> | <правило unconditional1y>
\ <правило when> | <правило whenever>]
\ <правило initially>]

<префікс for> ::= **for {any | the }** *<item> ...*

<правило if> ::= if *<логічне вираження>* **then** *<список дій> <правило*

*unconditionally> ::=***unconditionally** *<список дій>*

<правило when> ::= when *<логічне вираження>*
then *<список дій>*

<правило whenever> ::= whenever *<опис події>*
|or *<опис події>*
|and when *<логічне вираження>*

<правило initially> ::= **initially**
|if *<логічне вираження>* ***then*** *<список дій>*

Кожний з типів правил може бути як *загальним,* тобто стосовним до всього класу, так і *спеціалізованим,* стосовним до конкретних екземплярів класу. Можливість представляти знання у виді загальних, а не тільки конкретних правил, забезпечує наступні переваги:

- мінімізується надмірність БЗ;
- спрощується наповнення БЗ і її супровід;

- мінімізуються помилки при налагодженні БЗ;
- сприяє повторному використанню знань (тому що загальні правила запам'ятовуються в бібліотеці G2 і можуть використовуватися в подібних додатках).

Незважаючи на те, що продукційні правила забезпечують достатню гнучкість для опису реакцій системи на зміни навколишнього світу, у деяких випадках, коли необхідно виконати тверду послідовність дій (наприклад, запуск чи зупинку комплексу устаткування), більш кращим є процедурний підхід. Мова програмування, що використовується у G2 для представлення процедурних знань, є досить близьким родичем Паскаля. Крім стандартних керуючих конструкцій мова розширена елементами, що враховують роботу процедури в реальному часі: очікування подій, дозвіл іншим задачам переривати виконання даної процедури, директиви, що задають послідовне чи рівнобіжне виконання операторів. Ще одна цікава особливість мови - ітератори, *що* дозволяють організувати цикл над безліччю екземплярів класу. Перераховані властивості мови дозволяють системі одночасно виконувати безліч різних процедур чи безліч копій однієї і тієї ж процедури для безлічі різних об'єктів.

8.7. Машина виводу

Одним з основних компонентів G2 є машина виводу, що виконує міркування на підставі:

- знань, що містяться в базі знань;
- даних, що надходять від підсистеми імітаційного моделювання;
- даних, що надходять від зовнішніх джерел (контрольно-вимірювальної апаратури, СУБД і т.п.).

Правила збуджуються машиною виводу. При цьому, перевіряється істинність умови, що знаходиться в антецеденті правила. Якщо воно істинно, то машина виводу виконує дії, що знаходяться в консеквенті. При перевірці умови правила, машині виводу необхідно знайти значення всіх змінних і параметрів, що містяться

в них. Параметри в будь-який момент часу мають визначене значення, у той час як значення змінних може бути відсутнім, оскільки для них визначений час життя. Якщо змінна не має значення, система може одержати її значення з наступних джерел:

- сервер даних, що одержує дані від зовнішніх джерел (контрольно-вимірювальна апаратура, СУБД, інша ЕС і т.п.);
- оператор;
- підсистема моделювання зовнішнього оточення;
- правило, що визначає шукані значення змінних;
- формула, що приписана до змінної (класу чи змінних), значення якої потрібно системі.

Після того як усі необхідні значення отримані, система визначає, чи істинна умова розглянутого правила. Якщо умови правила дійсні, система виконує дії, що знаходяться в консеквенте даного правила. У набір дій, що використовуються в консеквентах правил, входять:

- присвоєння значення простому атрибуту, параметру чи змінній;
- посилка керуючої інформації зовнішньому об'єкту;
- запуск процедури;
- створення екземпляра об'єкта;
- видалення екземпляра об'єкта;
- породження і видалення задач (підзадач);
- зміна положення чи кольору піктограм на екрані дисплея;
- керування способом відображення робочих просторів (положення на екрані, масштаб і т.д.);
- вивод повідомлень для оператора системи;
- активізація всіх правил, асоційованих із заданим об'єктом;
- зупинка системи і т.д.

У зв'язку з тим, що G2 орієнтована на динамічні додатки, що працюють у реальному часі, машина виводу повинна мати засоби для скорочення перебору, реакції на непередбачені події і т.п.

Головним недоліком традиційно використовуваного в статичних ЕС прямого і зворотного виводує непередбачуваність витрат часу на їхнє виконання. Для динамічних систем с прямим і зворотнім виводом з повним перебором можливих до застосування правил - недозволенна розкіш. Наступна образна інтерпретація дозволяє зрозуміти недолік традиційних методів побудови ланцюжків логічного виводу і необхідність виходу на метарівень (focus і invoke) у динамічних системах. Уявіть собі, що ви прийшли в бібліотеку і хочете установити деякий факт. Дотримуючись алгоритму прямого виводу, ви повинні почати читати всі книги підряд за абеткою, доки не наткнетеся на потрібний факт. Коли факт установлений, і ваше знання про дійсність змінилося, дотримуючись того ж алгоритму, ви повинні спочатку перечитати всі книги, навіть уже прочитані!

Особливістю машини виводу G2 є багатий набір способів порушення правил. Правило в G2 може збуджуватися одним з 9 випадків. Методи 4-9 збуджують правило "при виникненні деякої події. Для реалізації цих методів у G2 введений спеціальний тип правил, що починаються з ключового слова whenever (як тільки). Правила whenever збуджуються в першу чергу і мають найбільший пріоритет, що гарантує своєчасну реакцію системи на зміни в навколишньому світі. Правила цього типу не використовуються (за замовчуванням) ні в прямому, ні в зворотному виводі, вони є метаправилами і реагують на події (переміщення об'єкта, установлення/усунення відносин, одержання/неотримання значення).

8.8. Планувальник

У зв'язку з тим, що С2-додаток керує безліччю одночасно виникаючих задач, необхідний Планувальник. Планувальник керує всіма процесами в G2 (Малюнок 8.1).Планувальник визначає порядок обробки задач, взаємодіє з джерелами даних і користувачами, запускає процеси і здійснює комунікацію з іншими процесами.

Планувальник циклічно виконує наступну послідовність кроків.

1. Перевірка початку циклу: якщо початок циклу наступив, планувальник починає цикл і переходить до наступного кроку.

2. Планування задач, що очікують: Планувальник формує список задач, що будуть виконаються на даному циклі - черга поточних задач.

3. Обслуговування джерел даних. Кожному джерелу даних приділяється не більш 0,1с на виконання цієї операції. Для джерел, що не закінчили обмін за виділений час, плануються задачі для спроби закінчити передачу даних.

Малюнок 8.1 Робота планувальника G2

Виконання задач: Планувальник бере чергу поточних задач і намагається виконати як можна більшу їх кількість. Кожна з задач, що не закінчилася протягом 0,2с, відкладається для виконання наприкінці даного циклу чи в наступному циклі.

4. Обслуговування мережних пакетів: Планувальник посилає й одержує повідомлення через мережу. На це виділяється до 0,2 с.

5. Обслуговування користувачів: Планувальник приймає і передає дані для всіх користувачів, що працюють у даному сеансі G2. Це включає і користувачів Telewindows.

6. Підготовка до наступного циклу: Планувальник перевіряє, чи залишилася будьяка активність у рамках даного циклу. Якщо так, (одержання даних, завершення відкладених задач, і т.п.), він повертається до кроку 1 і перевіряє, чи не наступив час нового циклу. Якщо ні, то переходить до кроку 3 для завершення усіх

відкладених задач. Якщо так, переходить на наступний цикл. Якщо відкладених задач не залишилось і час нового циклу не наступив, то настає пауза на 40 мс, після чого відбувається перехід до кроку 1.

8.9. Моделювання

Одним з можливих джерел даних для G2 є система моделювання зовнішнього оточення. Дана система використовується для моделювання реальних об'єктів і пристроїв, з якими працює ЕС. У системі передбачені наступні основні можливості:

- Засоби для обчислення алгебраїчних, різницевих рівнянь і диференціальних числень першого порядку;
- Можливість завдання формул як для окремих змінних, так і для класів змінних чи параметрів;
- Можливість режиму поділу часу, при якому моделювання працювало б паралельно з іншими підсистемами G2. За рахунок цього здійснюється обчислення модулюємих значень у процесі роботи механізму виводу.

Підсистема моделювання є досить автономною, але найважливішою частиною системи. На різних етапах життєвого циклу прикладної системи воно служить досягненню різних цілей. Під час розробки підсистема моделювання використовується замість об'єктів реального світу для імітації показань датчиків. Очевидно, що проводити налагодження на реальних об'єктах може виявитися занадто дорого, а іноді і небезпечно.

На етапі експлуатації прикладної системи процедури моделювання виконуються паралельно функціям моніторингу і керування процесом, що забезпечує наступні можливості:

- верифікацію показань датчиків під час виконання додатка;
- підстановку модельних значень змінної при неможливості одержання реальних (вихід з ладу датчиків чи тривалий час одержання відповіді на запит);
- Граючи роль самостійного агента знань, підсистема моделювання підвищує життєздатність і надійність додатків на базі G2. У G2 для опису зовнішнього світу

використовуються алгебраїчні, різницеві рівняння і диференціальні числення першого порядку;

Виділяють три типи змінних які можуть одержувати свої значення від підсистеми моделювання: *безупинні, дискретні* і *залежні*. Значеннями двох перших типів є функціями їхній попередніх значень, унаслідок чого для них повинні бути задані деякі початкові значення. З іншого боку, значення залежних змінних є функціями тільки поточних значень інших обчислюва змінних. Ця категорія змінних явно не з'являється; їхні значення виходять з рівнянь моделювання для відповідної змінної.

Прикладом використання рівнянь для обчислення обсягу деякого об'єкта.

Якщо ідентифікатором даного об'єкта є *бак_1* то алгебраїчною формулою для обчислення обсягу буде вираження:

*обсяг бак_1=рівень бак_1*площа бак_1,*

де *рівень і площа*- атрибути об'єкта *бак_1*.

Відповідне кінцево-різницеве рівняння для *обсягу бак_1* може мати вид:

Наступне значення обсяг бак_1=рівень бак_1+ 3

с початковим значенням 5000.

Диференціальне рівняння для цієї ж змінної *обсяг бак_1* виглядає так:

d/dt(обсяг бак 1) = (приплив на вході бак1-відтік на виході бак1)

с початковим значенням 5000.

Підсистема моделювання, як правило, забезпечує можливість завдання формул моделювання не тільки для окремих змінних і параметрів, але і для їхніх класів. За рахунок цього стає можливим однократний опис поводження, застосовуваний для всіх екземплярів класу. Вона дозволяє мати також різні збільшення часу при обчисленні різних змінних.

Як приклад нижче приведена загальна формула моделювання для обсягу будь-якого водяного бака :

d/dt (обсяг будь-який водяник_бак) = (приплив на вході будь-який водяник_бак -відтік на виході будь-який водяник_бак) з початковим значенням 5000.

Кожен параметр, значення якого задаються підсистемою моделювання, має атрибут алгоритм інтегрування, що визначає метод чисельного наближеного рішення звичайних диференціальних рівнянь. У G2 для цього використовують методи Эйлера і Рунге-Кутта. Вибір серед цих двох методів залежить від додатка.

Метод Рунге-Кутта дає більш точні результати, але вимагає в чотири рази більше витрат часу, чим метод Эйлера. Щоб досягти тієї ж точності, як у методі Рунге-Кутта, при використанні методу Эйлера необхідно значно зменшити час збільшення. Таким чином, для додатків, що вимагають точності і не пред'являють високих вимог до швидкості, ефективно використовувати метод Рунге-Кутта. З іншого боку, для додатків, у яких істотна швидкість, а не точність, доцільніше застосовувати метод Эйлера.

Стійкість (збіжність) методу не є фактором, на підставі якого варто вибирати той чи інший метод, тому що і метод Эйлера, і метод Рунге-Кутта мають туж саму межу стійкості щодо розміру кроку збільшення. Іншими словами, максимальний час збільшення, поза межами якого обчислення стають хитливими, теж саме для обох методів.

Середовище розроблювача в системі G2

Середовище розроблювача включає:

- Природно-мовний текстовий редактор, що керується процедурою граматичного розбору;
- інтерфейс із користувачем;
- засоби інспекції і налагодження;
- систему реєстрації версій.

8.10. Природно- мовний текстовий редактор

Розроблювач G2 надає інформацію про розроблювальний додаток на обмеженій англійській мові, і йому надана можливість посилатися на будь-яку сутність у БЗ багатьма способами. Наприклад, розроблювач може використовувати конструкції природної мови (ЕЯ) для того, щоб послатися на сутність такими способами.

1. По імені: pump-12 (насос-12).

2. За допомогою префікса "for" (для) і слова "any" (будь-який), за якими слідує ім'я класу. Таким способом забезпечується посилання не на одну сутність, а на групу сутностей:

for any pump (для будь-якого насоса)...

Це забезпечує можливість записувати загальні твердження, наприклад, такі:

for any tank
if the tank is empty
then inform the operator that "The tank ...is empty"

(для будь-якої цистерни, якщо ця цистерна порожня, то слід інформувати оператора, що "цистерна ... порожня").

3. Як на одну із сутностей класу об'єктів, зв'язаних з іншим об'єктом:
the tank connected to the pump (цистерна, зв'язана з насосом).

4. Як на об'єкт, найближчий на графічній моделі до деякого об'єкта:
the pump nearest to the tank (насос, найближчий до цистерни).

Незважаючи на складність і багатство синтаксичних конструкцій G2 для опису знань, їхнє застосування спрощується за рахунок природно-мовного підходу. Прикладом використання природної мови в G2 для формування бази знань може служити наступне типове правило:

"If the altitude of any aircraft < the safe-flying-altitude of

the aircraft then inform the operator that "Pull up. You are flying too low. Your altitude is [the altitude of the aircraft]"

Спрощення взаємодії розроблювача із системою досягається і за рахунок оригінального підходу, реалізованого в текстовому редакторі.

Інтерактивний текстовий редактор G2 дозволяє редагувати тексти тверджень, правил, функцій, процедур і т.д. Він працює в спеціальному вікні редагування, що з'являється, як тільки ініціюється створення нового твердження чи редагування існуючого, вибирається будь-яка ділянка тексту, додається чи редагується будь-який інший текст, включаючи текст, що представляє значення атрибута екземпляра класу. Процес редагування увесь час направляється процедурою граматичного розбору, що гарантує введення тільки синтаксично правильних конструкцій мови. У вікні редагування з'являється динамічно змінювана підказка, що вказує, які мовні конструкції можна вводити, починаючи з поточної позиції курсору. Існує можливість набирати текст, що вводиться, на клавіатурі чи вибирати придатні шаблони з підказки. Крім того, для спрощення редагування можна використовувати клавіатурні команди чи контекстно-залежне меню операцій редагування. Наприклад, правило, приведене вище (крім тексту, що укладений у лапк), може бути введене за допомогою 16 натискань клавіші мишки і введення з клавіатури трьох букв A и однієї букви S.

Так само, як немає необхідності заучувати напам'ять граматичні конструкції мови для написання правил і процедур G2, не потрібно вивчати і мову графічних примітивів для побудови піктограм об'єктів. Редактор піктограм дозволяє створювати піктограми графічними засобами й автоматично перетворювати їх у текстові описи. У результаті можна бачити, як буде виглядати піктограма, і змінювати її.

Нижче приводиться короткий перелік основних можливостей графічного інтерфейсу:

- використання растрової графіки поряд з векторною для піктограм об'єктів і фонових зображень;

- різні типи графіків і зручний інтерфейс для їх конфігурування;
- можливість вибору між "шаровим" (що перекривається) чи "напівпрозорим" (XOR) промальовуванням елементів інтерфейсу;
- довільне масштабування графічних елементів;
- різноманітні векторні шрифти;
- підтримка формату Enhanced PostScript при збереженні зображення у файлі для наступної розпечатки;
- різноманітні способи роботи з меню, текстовими і графічними редакторами;
- планована на основі пріоритетів, що задаються користувачем, перемальовування окремих ділянок екрана, що гарантує першочергове відновлення найбільш важливої інформації на графіках, у діаграмах, таблицях і т.п.;
- різноманітні функції обробки клавіатури і миші, що дозволяють використовувати спеціалізовані інтерфейсні рішення при організації робочого місця оператора;
- підтримка стандарту ISO 8859-5 у частині представлення символів кирилиці незалежно від операційного середовища. Ця особливість відкриває російськомовним розроблювачам можливість використання російських імен у назвах класів, атрибутів і т.д.

Кінцевий користувач може взаємодіяти з G2 різними способами. Так, у G2 розроблювач створює різноманітні меню, що дають кінцевому користувачу наступні можливості:

- показати (сховати) робочий простір;
- рухати чи обертати сутність;
- змінювати кольору "ікон" і зв'язків;
- видавати повідомлення і т.д.

У додавання до цих засобів взаємодії G2 надає кінцевому користувачу наступні засоби:

- зображення (displays);

• керуючого впливу на G2 (end-user controls);

• повідомлення;

• керування доступом (access control);

• створення опцій меню (user menu choices);

• перекладу опцій меню.

8.11. Зображення

Зображення використовуються для того, щоб надати користувачу можливість побачити значення змінних і виражень. Існують наступні варіанти, що реалізують можливість зобразити (display):

• Readout-table (відлік). Показує (у горизонтальному прямокутнику) ліворуч ім'я змінної (параметра) чи вид вираження, а праворуч - значення.

• Dial (циферблат). Зображує арифметичне значення у виді крапки на круговій шкалі. При русі по годинній стрілці значення збільшується.

• Meter (вимірник). Зображує у виді вертикальної шкали з покажчиком значення арифметичного вираження, змінної (параметра).

• Graph (графік). Зображує в двох вимірах зміну одного чи більш виражень (в окремому випадку параметра чи змінної) у часі.

• Chart (діаграма). Відображає на графіку (із двома ортогональними осями) співвіднесення однієї послідовності даних з деякою іншою послідовністю.

•Freeform-table (таблиця). Зображує таблицю з рядів і рядків (подібна електронній таблиці). Значення кожного осередку в таблиці може обчислюватися G2.

G2 дозволяє реалізувати можливість display динамічно (тобто в процесі роботи механізму виводу) за допомогою дії create (створити).

8.12. Керуючі впливи

Керуючі впливи (end-user controls) - це засоби, за допомогою яких кінцевий користувач може взаємодіяти з додатком. Існують наступні види керуючих впливів:

- Action button (кнопка дії). Зображується у виді прямокутника з закругленими кутами. Натискання розроблювачем на кнопку дії приводить до виконання зв'язаних з нею дій. Наприклад, таких, як: change (змінити); conclude (укласти); create (створити); delete (усунути); halt (зупинити); hide (сховати); inform(інформувати); move (рухати); rotate (обертати); transfer (перенести) і т.п.

- Radio button. Зображується у виді групи маленьких кругових "іконок", що відповідають взаємовиключній безлічі символів, чисел, логічних чи текстових значень. При натисканні на одну з "іконок" відповідне їй значення привласнюється змінній чи параметру. Наприклад, користувач може використовувати групу radio buttons, позначену red (червоний), black (чорний), white (білий), для призначення кольору деякої символічної змінної.

- Check boxes (кнопка перевірки). Зображується у виді маленького квадрата разом зі зв'язаним з ним значенням змінної чи параметра (символічним, кількісним, логічним, текстовим).Check boxes перевіряє, чи є зазначене значення активним. Якщо значення активне (on), то в квадраті Check boxes з'являється хрест, якщо значення неактивне (off, тобто якесь інше значення), в квадраті нема нічого; якщо значення не відоме, то в квадраті "?" (знак питання).

- Slider (покажчик). Зображується у виді горизонтальної шкали з покажчиком значення для початку і кінця покажчика. Переміщаючи покажчик уздовж горизонтальної шкали, користувач може вводити числове значення змінної чи параметра.

- Type-in box (уведення). Зображується у виді прямокутника. Значення (числове чи текстове) змінної чи параметра, зв'язане з Type-in box, може бути введене в прямокутник користувачем із клавіатури. Якщо яке-небудь інше джерело (не Type-in box) змінить значення змінної (параметра), то Type-in box відобразить ця зміна.

8.13. Повідомлення

Повідомлення (messages) є клас сутностей, що містять у собі текст. Повідомлення є засобом, що дозволяє G2 інформувати користувача про які-небудь події. Наприклад, у результаті виконання твердження inform (інформувати) G2 створює деяке повідомлення і розміщає його на дошці повідомлень (message-board); при виявленні деякої помилки G2 видає повідомлення в журнал оператора (the operator logbook).

Повідомлення, що G2 створює в результаті виконання дії inform, чи повідомлення як реакція на помилку і т.п., є прикладами вбудованого класу сутностей, що називається message (повідомлення). Розроблювач може створити підклас класу message, що буде мати унікальні характеристики й атрибути. Наприклад, клас повідомлень user-warning-message (попереджуючі користувальницькі повідомлення), може використовувати текст із дуже великим шрифтом, розміщеним на тлі заданого кольору. Повідомлення стандартно видаються на одину з двох робочих просторів: на дошку повідомлень (message board) чи журнал оператора (logbook). Звичайно на дошку повідомлень видаються повідомлення для користувача, що викликаються дією inform. Вид дошки повідомлень керується за допомогою атрибута message-board-parameters у системній таблиці (system tables). Повідомлення про помилки, системні умови і попередження видаються в журнал оператора. Вид журналу керується через атрибут logbook-parameters у system tables.

8.14. Керування доступом

За допомогою засобів керування доступом (access control) розроблювач може впливати на те, що кінцевий користувач бачить і може робити з БЗ. Наприклад, розроблювач може керувати доступом у такий спосіб:

• обмежити (restrict) рядки меню, що бачить користувач у кожнім меню;

• обмежити користувачів у можливості пересувати сутності, установлювати зв'язки і т.п.;

• визначити перелік атрибутів, що може бачити користувач у таблицях атрибутів, у робочих просторах, в об'єктах і т.д.;

• дозволити користувачу бачити атрибути сутностей, але не редагувати їхній і т.п.;

• забезпечити автоматичне виконання деякої дії, наприклад, показати робочий простір об'єкта, коли користувач указує на сутність.

Обмеження (restrictions), що призначає розроблювач, можуть діяти:

• на всі сутності в БЗ;

• на визначені класи сутностей;

• на сутності визначеного робочого простору;

• на приватні сутності.

Розроблювач керує доступом за допомогою вказівки типу користувача (mode), що працює з додатком: оператор, розроблювач, адміністратор і т.п. Розроблювач може розширити список типів користувачів за своїм розсудом. Тип користувача "адміністратор" вбудований у G2, цей тип користувача вільний від всіх обмежень, він може бачити і робити все, на що здатна G2.

8.15. Створення опцій меню

Розроблювач може визначити нові опції (рядка) меню понад тих, що використовуються стандартно. Коли користувач вибирає опцію меню (user menu choise - umc) для того, щоб внести новий рядок у меню, що відповідає приватному класу сутностей, G2 виконує визначені дії. Додатковий рядок у меню з'являється для прикладів відповідних сутностей під час виконання БЗ і при дотриманні умов, зазначених у umc.

Наприклад, припустимо, що БЗ містить клас об'єктів, називаний "перемикач", і цей клас має атрибут "стан" зі значенням "включене" (on) чи "виключено" (off). Розроблювач може за допомогою umc додати рядок до меню "перемикач", що у стані перемикача on буде містити в меню рядок off і навпаки.

8.16. Переклад опцій меню

G2 дозволяє перевести імена будь-яких опцій меню з англійського на іншу мову. Це позначає, що розроблювач G2 може замінити деякі опції з однієї мови на іншій. Наприклад,

in Ukreine:
table = таблиця
move = перемістити
edit = редагувати.

8.17. Засоби інспекції і налагодження

Ясно, що налагодження прикладної системи, що поєднує правила, процедури, різні рівні абстракції й ієрархію класів, може перетворитися в далеко не тривіальну задачу. У цій ситуації на допомогу розроблювачу приходять могутні засоби інспекції бази знань і налагодження, надані G2. Функції інспекції бази знань дозволяють здійснювати пошук елементів на основі їхніх типів, приналежності до класу, атрибутів і місця розташування. Ці функції використовуються для рішення наступних задач:

- відображення стиснутого представлення елементів бази знань;
- створення файлів, що містять опис елементів бази знань;
- відображення ієрархій класів, модулів і робочих просторів;
- прямий перехід до конкретних елементів бази знань;

Зокрема, вище згадана схема ієрархії вбудованих класів G2 може бути отримана за допомогою функції інспекції бази знань.

Перераховані можливості полегшують навігацію по базі знань і надають можливість швидкого перегляду бази знань під будь-яким ракурсом. Крім того, за допомогою функцій інспекції можна запустити процедуру пошуку і заміни текстових фрагментів у базі знань. Функції інспекції працюють у фоновому режимі і

дозволяють виконувати одночасно з ними й інші задачі, включаючи інші функції інспекції. Доступ інших користувачів до бази знань у цей час ніяк не обмежується.

8.18. Інтерфейс із зовнішнім оточенням

У G2 реалізована розподілена обробка додатка на принципах архітектури клієнт-сервер. Клієнтна система Telewindows забезпечує множинний доступ до централізованої бази знань і групову роботу з додатком. Взаємодія між G2 і Telewindows може бути організована одним з наступних способів: процес Telewindows виповнюється на тієї ж машині, що і G2, а користувач одержує до неї доступ через X-термінал; Telewindows виповнюється на робочій станції чи ПК користувача. Крім того, додаток можна побудувати як співдружність автономних інтелектуальних агентів на базі інтерфейсу G2↔G2, що виконуються на одній і тієї ж чи на різних ЕОМ, зв'язаних у мережу. При цьому обмін даними здійснюється на рівні змінних через протокол ICP (Intelligent Communication Protocol). Інтерфейс G2↔G2 дозволяє розроблювачу створити в одному G2-додатку об'єкти, що одержують інформацію від іншого G2. Ці об'єкти створюються, подібно іншим об'єктам у G2, а інтерфейс G2↔G2 діє як сервер даних для цих об'єктів (подібно механізму виводу чи G2-підсистемі моделювання). Для організації обміну необхідно в описі змінних об'єкта, що одержують значення від іншого G2-процесу, просто вказати номер мережного порту джерела. Як результат, змінні об'єкта одержать значення від другого G2. Підкреслимо, що G2-додаток може як одержувати, так і посилати інформацію в один і той ж час по одному інтерфейсу.

G2 розроблена як відкрита система. Зв'язок із зовнішніми джерелами даних будується на основі бібліотеки стандартних інтерфейсів і сервера GSI (G2 Standart Interface). Підсистема GSI працює паралельно з прикладною системою як незалежний оброблювач подій і забезпечує її двосторонню взаємодію із широким спектром програмувальних контролерів ведучих фірм (Allen Bradley, GE-Fanuc, AEG Modicon), систем збору даних (ABB, Fisher, Siemens, Yokogawa, Foxboro, ORSI), концентраторів даних (DEC BASEstar, Allen Bradley Pyrammid Integrator, SETPOINT SETCIM) і розвитих СУБД (Oracle, Sybase, DEC Rdb). Бібліотека GSI і G2 Bridge

products дозволяють легко інтегрувати G2-додаток в існуючі системи керування. При відсутності в бібліотеці GSI інтерфейсу до деякого унікального контролера не важко запрограмувати його по шаблону на мові С і підключити до системи.

З погляду сучасної концепції розробки відкритих систем у системі G2 пропонується більш гнучка і надійна трикомпонентна схема організації взаємодії клієнтської і серверної частин додатка, використовуючи GSI як монітор транзакцій.

Звичайно, коли перед розроблювачем постає проблема створення інтерфейсу даних, він змушений брати до уваги цілий ряд різнорідних вимог: ефективність, реалізуємість, надійність, переносимість, супроводжуємість, конфігураційність, гнучкість, можливість мультиплексування сигналів і т.д. Для задоволення цих вимог він повинен реалізувати безліч функціональних блоків, таких, як: синхронізація оброблюваних запитів, протоколи взаємодії; мережні інтерфейси, відновлення після збоїв у мережі чи вузлі; робота з безліччю джерел даних; угруповання даних; обробка даних, що прийшли без запиту; обмін повідомленнями про помилки; обмін даними про стан взаємодіючих систем; буферізація даних; перетворення форматів даних, робота з даними змінної довжини; планування обробки запитів; відпрацьовування запуску і зупинки системи; відпрацьовування пауз і перезапуску; відпрацьовування переривань; обробка переповнення буфера; розподіл ресурсів; мінімізація завантаження системи; діагностика збоїв; доступ до зовнішніх даних і конфігурування інтерфейсу. Усе це застосовно до будь-якого інтерфейсу даних незалежно від призначення прикладної системи. Усі перераховані вимоги, крім двох, задовольняються в підсистемі GSI автоматично незалежно від платформи і типу мережного забезпечення. Виключення складають функції доступу до даних і конфігурування інтерфейсу, але реалізацію цих функцій GSI робить настільки простою, наскільки це можливо.

Підсистема GSI складається з трьох основних частин:

• ядро GSI;

• GSI- розширення;

• комунікаційний канал зв'язку між ядром GSI і GSI розширенням.

Хоча підсистема GSI відпрацьовує усі взаємодії між G2 і кожним із зовнішніх процесів, необхідно сконфігурувати її для конкретного додатка. Для розробки системи, у повному обсязі можливості, що використовує GSI, потрібно створити два

фрагменти, що відбивають специфіку прикладної програми, у додаток до бази знань G2: специфікацію конфігурування, що настроїть базу знань для зв'язку з зовнішньою програмою, і так називаний перехідний код (application bridge code), що використовується GSI-розширенням для інтерактивної взаємодії з зовнішньою прикладною програмою.

Специфікація конфігурування включає об'єкти бази знань, що конфігурують її для використання GSI. Засоби для створення цих об'єктів вбудовані в G2. Для формування специфікації конфігурування створюють об'єкти, що належать до до GSI interface object. Ці об'єкти містять інформацію, необхідну GSI для зв'язку з зовнішньою прикладною програмою. Далі створюють змінні класу GSI variable, що відповідають змінним зовнішньої прикладної програми. G2 передає і приймає ці змінні. Крім цього створюють змінні GSI variable для обміну текстовими повідомленнями з зовнішньою прикладною програмою.

Перехідний код поєднує GSI-розширення з зовнішньої при ній програмою. Він складається з набору функцій мовою С, що забезпечують передачу даних, текстових повідомлень, запуск, зупинку і завершення зовнішньої програми.

Крім інтерфейсів GSI і G2↔G2 доступні ще два інтерфейси з зовнішніми процесами і джерелами даних: файловий інтерфейс (G2 File Interface - GFI) і інтерфейс із зовнішніми функціями (Foreing Function Interface).

Файловий інтерфейс GFI являє собою гнучкий засіб, що дозволяє G2 писати і читати інформацію з зовнішніх файлів. GFI є окремим від G2 продуктом. Розроблювач може використовувати GFI для того, щоб робити наступне:

- архівірувати і запам'ятовувати дані;
- ініціалізувати тести перевірки БЗ;
- збирати дані для зовнішнього аналізу;
- створювати знімки (snapshots) даних;
- зчитувати дані з зовнішніх файлів під час виконання.

Інтерфейс із зовнішніми функціями. Розроблювач G2-додатка може викликати зовнішні (foreign) для G2 функції, написані на С і Фортрані. Цей інтерфейс включений у G2. Для того щоб використовувати в G2 зовнішню функцію,

розроблювач описує її і потім використовує таким же способом, як і функції, визначені користувачем.

Відкритість системи G2 і продуктів на її основі забезпечується орієнтацією фірми Gensym на промислові стандарти. Будучи членом OMG (Object Management Group), фірма Gensym співробітничає в цій області з багатьма незалежними організаціями і комітетами зі стандартів. Стосовно технічних засобів - це підтримка широкого спектра платформ DEC, HP, Sun, IBM і ПЕОМ на базі процесорів X86 і Pentium. Розвитий графічний інтерфейс, що включає елементи анімації, базується на засобах Motif і MS Windows. Мережні протоколи - TCP/IP і DECnet. Архітектура клієнт/сервер на рівні обміну даними підтримується монітором транзакцій GSI і DDE, на рівні об'єктів - CORBA, на рівні додатка - клієнтною підсистемою Telewindows. Розподілена обробка забезпечується інтерфейсами G2↔G2, G2 - Telewindows і підтримкою виклику вилучених процедур. Існує безліч готових рішень "під ключ" для прямої взаємодії G2 з розповсюдженими програмними і технічними системами контролю і розвитими СУБД.

Можливість простого маніпулювання графічним представленням об'єктів у G2 і складання схем, що є відображенням технологічних ланцюжків чи абстрактних алгоритмів обробки даних, забезпечує базові засоби для побудови проблемно-орієнтованих мов візуального програмування. У цьому випадку об'єкти набувають властивості операторів і в сукупності з різними класами зв'язків формують граматику нової мови. Основною перевагою такого підходу є те, що сформована діаграма потоків інформації по суті і є програма, що виконується, проміжні фази генерації коду і компіляції для її використання не вимагаються. Уперше ця концепція була реалізована фірмою Gensym у GDFL- мові графічного представлення інформаційних потоків для побудови систем діагностики реального часу в системі GDA (G2 Diagnostic Assistant). Підхід виявився настільки вдалим, що тією чи іншою мірою використовується тепер у всіх проблемно/предметно-орієнтованих розширеннях G2.

Крім базового продукту - оболонки ЕС реального часу G2, на її основі фірмою Gensym розроблені додаткові проблемно-орієнтовані засоби розробки; основні з

них: G2 Diagnostic Assistant (GDA) – середовище візуального програмування на базі G2 для створення додатків активної діагностики виробничих та інформаційних процесів; NeurOn-Line (NOL)- середовище, що інтегрує технологію нейронних мереж і експертних систем; ReThink- середовище моделювання і реінжинірінгу діяльності компаній.

9. Інформаційні - ресурси Інтернет

Нижче надані деякі ресурси Інтернет, присвячені проблемам штучного інтелекту й експертних систем:

1. http://www.aaai.org – сервер Американської асоціації штучного інтелекту (American Association for Artificial Intelligence).

2. http://www.ai-cbr.org/theindex.html – методика використання прецендентів у системах штучного інтелекту.

3. http://www.comlab.ox.ac.uk/archive/comp/ai.html – каталоги деяких лабораторій із проблем штучного інтелекту.

4. http://www.cs.cmu.edu/Groups/AI/html/repository.html – сховище публікацій зв`язаних з штучним інтелектом при університеті Карнегі - Меллона.

5. http://www.cs/washington.edu/research/jair/home.html – електронний журнал “Journal of Artificial Intelligence Research”.

Додаток 1

```
% чи     Існує зв'язок між X  і  Y ?
      domains
                  city=string
       %  Опис предикатів
      predicates
                  dconnect(city,city)
                  connect(city,city)
                  wopros
      % База знань
      clauses
      %  Факти, що описують дерево
                  dconnect (a,b).     dconnect (a,c).    dconnect (b,d).     dconnect (b,e).
                  dconnect (c,f).      dconnect (c,q).    dconnect (f,h).     dconnect (f,i).
      % Правила виводу
                  connect(X,Y) if dconnect(X,Y).
                  connect(X,Y) if  dconnect(X,Z), dconnect(Z,Y).
      wopros:-
                  write(" Введіть від - "), readln(X),
                  write(" Введіть до - "), readln(Y), connect(X,Y),nl,
                  write (" від ",X," до ",Y," є зв`язок "), nl,!.
      wopros:- write("  НЕМАЄ РІШЕНЬ!").
      % Внутрішня мета
      goal
                  makewindow(1,120,7," відповідь ",16,0,8,60),
                  wopros.
```

Додаток 2

```
DOMAINS
toch=string
        km,ml=real
        Int=integer
        slist=string*
        sslist=slist*
        rlist=real*

/* Основні та службові бази даних */

DATABASE - sosed
                    sosed(toch,toch,km,ml)

DATABASE - lists
                    lists(slist)
DATABASE - sum
                    sum(real)

PREDICATES
                connect(toch,toch,km,ml,slist,slist)
                wopros
               start
               autoload
                working(Int)
               main_menu
                par(toch,toch,Int)
                repeat
                find(string,slist)
```

```
        add(slist,string,slist)
        result(slist,slist,slist)
        adds(slist)
        writelist(slist)
        min(rlist,real,integer,integer,real,integer)
        createSum(real,real)
        summa(slist,real,real,real,real)
        dcon(toch,toch,real,real)
        createListR(rlist,rlist)
        addr(rlist,real,rlist)
        createListS(sslist,sslist)
        addss(sslist,slist,sslist)
        first(rlist,real)
        writeN(sslist,integer)
        writeall(sslist)
GOAL start.
CLAUSES

    writelist([]).
    writelist([X|L]):-write(" ",X), writelist(L).

            dcon(X,Y,M,N) :- sosed(X,Y,M,N);sosed(Y,X,M,N).

            summa([_],N,S,_,_) :- S=N.
            summa([X,Y|L],N,S,Км,Бензин) :-bound(Км),dcon(X,Y,M,_),
K=M+N, summa([Y|L],K,S,Км,Бензин);
                    bound(Бензин), dcon(X,Y,_,M), K=M+N,
summa([Y|L],K,S,Км,Бензин).

            add(L1,S,[S|L1]).
```

```
adds(X) :- lists(X), !.
adds(X) :- assertz(lists(X)).

createSum(Км,Бензин) :- lists(X), summa(X,0,Suma,Км,Бензин),
assertz(sum(Suma)), fail.
createSum(_,_).
addr(L1,R,[R|L1]).
createListR(R,L) :- not (sum(_)), R=L.
createListR(R,L) :- sum(X),
addr(L,X,M),retract(sum(X)),createListR(R,M).

addss(L1,R,[R|L1]).
createListS(R,L) :- not (lists(_)), R=L.
createListS(R,L) :- lists(X), addss(L,X,M), retract(lists(X)),
        createListS(R,M).

first([X|_],N) :- N=X.

min([],Tmp,Cnt,_,Min,N) :- Min=Tmp, N=Cnt.
min([X|L],Tmp,Cnt,K,Min,N) :- Tmp>X, C=K, KK=K+1, MM=X,
 min(L,MM,C,KK,Min,N);
 KK=K+1, min(L,Tmp,Cnt,KK,Min,N).

writeN([L|_],0) :- writelist(L).
writeN([_|L],N) :- M=N-1, writeN(L,M).
find(R,[R|_]).
find(R,[_|T]) :- find(R,T).

result([_],L,R) :- R=L.
result([X|L],M,R) :- add(M,X,L1), result(L,L1,R).
```

```
writeall([]).
writeall([X|L]) :- writelist(X), nl, writeall(L).

/* Основна  процедура start */
start:-
        autoload,
        main_menu.

% Процедура connect

connect(X,Y,_,_,M,R) :-
        sosed(X,Y,AA,BB), R=M;
        sosed(Y,X,AA,BB), R=M.

connect(X,Y,Км,Бензин,L,R) :-
  sosed(X,Z,_,_),
  not (find(Z,L)), add(L,Z,M), connect(Z,Y,Км,Бензин,M,R);
  sosed(Z,X,_,_),
  not (find(Z,L)), add(L,Z,M),
  connect(Z,Y,Км,Бензин,M,R).

par(X,Y,'1'):-removewindow(15,1),L=[X,Y],connect(X,Y,_,_,L,R),
        result(R,[Y],M), adds(M), fail.
par(_,_,'1'):- not(lists(_)), write("НЕМА ШЛЯХУ").
par(_,_,'1'):-createSum(0,_), createListR(L1,[]), first(L1,Z),
min(L1,Z,0,0,M,N),  write("Мінімальна відстань = ",M,"км."),createListS(R,[]),
        nl,write("Траєкторія шляху -  "),writeN(R,N).
par(X,Y,'2'):- removewindow(15,1),L=[X,Y],connect(X,Y,_,_,L,R),
                result(R,[Y],M), adds(M), fail.
par(_,_,'2'):- not(lists(_)), write("НЕМА ШЛЯХУ").
par(_,_,'2'):-createSum(_,0), createListR(L1,[]), first(L1,Z),
```

```
        min(L1,Z,0,0,M,N),  write("Мінімальні витрати бензину = ",M," мл."),
              createListS(R,[]),nl, write("  Траєкторія шляху - "),writeN(R,N).

        par(X,Y,'3'):- removewindow(15,1),L=[X,Y], connect(X,Y,_,_,L,R),
              result(R,[Y],M), adds(M),fail.
        par(_,_,'3'):- not(lists(_)),write("НЕМА РІШЕНЬ ").
        par(_,_,'3'):- write("Все маршруты :"),nl,createListS(R,[]),writeall(R).
% Процедура wopros
    wopros :-
              retractall(_,lists),
              retractall(_,sum),
              write("Шукаемо шлях з параметрами "),nl,
              write("От"),nl,readln(X),write("До"),nl,readln(Y),
           makewindow(15,$1E,$2F," ВВОД ПАРАМЕТРІВ ",6,18,9,40),
              write("       1   - Відстань"),nl,
              write("       2   - Витрати бензину"),nl,
              write("       3   - Усі шляхи"),nl,
              write("      ->"),
              readchar(N),
              clearwindow,
              par(X,Y,N),
              removewindow(15,0).
        wopros :-  removewindow(15,0).

/* Процедура автоматичного завантаження баз даних */
        autoload:-
                  retractall(_,sosed),
                  consult("pal.ddd",sosed).
        autoload:-
                  makewindow(2,74,79,"ПОМИЛКА",6,18,8,40),
                  cursor(2,10),
```

```
                write("Немає бези на диску"),
                sound(70,294),
                removewindow,
                !.

/* Головне меню */

        main_menu:-
                repeat,
        makewindow(1,$1E,$2F,"ТРАЄКТОРІЯ ШЛЯХУ РОЗНОЩИКА
                        ПІЦИ",0,0,25,80),cursor(7,0),

                write("          1 – Збереження  бази"),nl,
                write("          2 – Додавання пункту доставки"),nl,
                write("          3 - Видалення пункту доставки"),nl,
                write("          4 - Перегляд бази"),nl,
                write("          5 – Розв`язання задачі"),nl,
                write("         6 - Вихід з програми"),nl,
                write("                        ->"),
                readint(C),
                clearwindow,
                working(C),
                clearwindow,
                C = 6,
                retractall(_),
                removewindow(1,0).

/* Процедура збереження бази даних */
        working(1):-
                makewindow(5,$5F,$5F," ЗБЕРЕЖЕННЯ БАЗИ ",8,20,9,40),
                sound(5,220),
```

```
                repeat,
                save("pal.ddd",sosed),nl,nl,
                write("          База збережена"),nl,nl,nl,
                write("       Натисніть будь-яку клавішу..."),nl,
                readchar(_),
                clearwindow,
                !,
                removewindow.

/* Процедура Додавання зв`язків  */

    working(2):-
        makewindow(6,$1E,$2F," ДОДАВАННЯ ПУНКТУ ДОСТАВКИ ",0,0,25,80),
        sound(5,220),
        cursor(9,0),
        write(" введіть ім`я  1 пункту      ->"),readln(P1),
        write(" введіть ім`я 2 пункту      ->"),readln(P2),
        write(" введіть відстань  (у км.)      ->"),readreal(Км),
        write(" введіть витрати бензину (у мл.)  ->"),readint(Бензин),
        assertz(sosed(P1,P2,Км,Бензин)),
        clearwindow,!,
        removewindow(6,0).

/* Процедура Видалення зв`язків */
            working(3):-
            makewindow(7,26,$4F,"      ВИДАЛЕННЯ      ПУНКТУ      ДОСТАВКИ
",0,0,25,80),
                sound(5,220),
                cursor(9,0),
                write("введіть ім`я  1 пункту      ->"),nl,
                readln(P1),
```

```
                write("введіть ім`я  2 пункту      ->"),nl,readln(P2),
                retract(sosed(P1,P2,_,_)),
                clearwindow,
                !,removewindow.

  /* Процедури перегляду баз даних */

      working(4):-
            makewindow(9,$1E,$2F," ПЕРЕГЛЯД БАЗИ ДАНИХ ",0,0,25,80),nl,
            write("
г=============================================================¬"
),nl,
            write(" ¦                  База даних                 ¦"),nl,
            write("
¦==============T==============T============T====================¦")
,nl,
      write(" ¦ Пункт #1     ¦  Пункт  #2  ¦ Відстань ¦ Витрати бензину     ¦"),nl,
      write(" ¦              ¦             ¦          ¦                     ¦"),nl,
            write("
¦==============+==============+============+====================¦"),
nl,
            sosed(P1,P2,Км,Бензин),cursor(Z,_),
            cursor(Z,0),write(" ¦  ",P1),
            cursor(Z,15),write(" ¦  ",P2),
            cursor(Z,29),write(" ¦  ",Км),
            cursor(Z,42),write(" ¦  ",Бензин),
            cursor(Z,63),write(" ¦"),nl,fail.
      working(4):-
            write("
L==============¦==============¦============¦====================-"),
                nl,cursor(22,26),
```

```
                write("Натисніть будь-яку клавішу"),
                readchar(_),
                removewindow,!.

% Процедура розв`язання задачі
    working(5) if
                makewindow(10,26,48,"Розв`язання задачі",0,0,25,80),
                nl,nl,
          sound (5,220),wopros,sound(5,220),
                cursor(22,26),
                write("Натисніть будь-яку клавішу "),
                readchar(_),
                removewindow(10,1),!.

      /* Процедура виходу з програми */

          working(6):- removewindow(1,1).
          working(_):-
          makewindow(11,$4E,$4F,"ПОМИЛКА ВВОДУ",6,18,8,40),  nl,
          write("       Введіть число від 0 до 6,"),nl,
          write("  що відповідаєобраному пункту"),nl,nl,nl,
                write("       Натисніть будь-яку клавішу "),
                sound(20,494),
                sound(30,392),
                readchar(_),
                removewindow(11,1).

/* Процедура repeat */
          repeat.
          repeat:- repeat.
```

Додаток 3

```
% Експерт по породах собак. Продукційна система, що базується на правилах
domains
  database
            xpositive(symbol,symbol)
            xnegative(symbol,symbol)
  predicates
            do_expert_job
            do_consulting
            ask(symbol,symbol)
            dog_is(symbol)
            it_is(symbol)
            positive(symbol,symbol)
            negative(symbol,symbol)
            remember(symbol,symbol,symbol)
            clear_facts
  goal
            do_expert_job.
clauses
/* Система користувальницького інтерфейсу*/
   do_expert_job :-
            makewindow(1,7,7,"AN EXPERT SYSTEM",1,16,22,58),
            nl,write("* * * * * * * * * * * * * * * * * * * *"),
            nl,write("  WELCOME TO A DOG EXPERT SYSTEM       "),
            nl,write("                                       "),
            nl,write("This is a dog identification system.   "),
            nl,write("Please answer the question about       "),
            nl,write("the dog you would like by typing in    "),
            nl,write("'yes' or 'no'.                         "),
```

```
        nl,write("* * * * * * * * * * * * * * * * * * * *"), nl,nl,
        do_consulting,
        write("Press space bar."),nl,
        readln(_),
        removewindow, exit.
do_consulting :-
        dog_is(X),!,nl,
        write("the dog you have indicated is a(n)",X,"."),nl,
        clear_facts.
do_consulting :-
        nl,write("Sorry I can't help you ! "),
        clear_facts.
ask(X,Y) :-
        write("  Question :- ",X," it ",Y," ?"),
        readln(Reply),
        remember(X,Y,Reply).
      /*              МЕХАНІЗМ ВИВОДУ        */
      positive(X,Y) :-  xpositive(X,Y),!.
      positive(X,Y) :-
% not
        not( negative(X,Y)),!,
        ask(X,Y).
negative(X,Y) :-
        xnegative(X,Y),!.
remember(X,Y,yes) :-
        asserta(xpositive(X,Y)).
remember(X,Y,no) :-
        asserta(xnegative(X,Y)), fail.
clear_facts :-
        retract(xpositive(_,_)), fail.
clear_facts :-
```

```
        retract(xnegative(_,_)), fail.

    /*  ПРОДУКЦІЙНІ ПРАВИЛА          */
dog_is("English Bulldog") :-
        it_is("short-haired dog"),
        positive(has,"height under 22 inches"),
        positive(has,"low-set tail"),
        positive(has,"good natured personality"),!.
dog_is("Beagle") :-
        it_is("short-haired dog"),
        positive(has,"height under 22 inches"),
        positive(has,"longer ears"),
        positive(has,"good natured personality"),!.
dog_is("Great Dane") :-
        it_is("short-haired dog"),
        positive(has,"low-set tail"),
        positive(has,"good natured personality"),
        positive(has,"weight over 100 lb"),!.
dog_is("American Foxhound") :-
        it_is("short-haired dog"),
        positive(has,"height under 30 inches"),
        positive(has,"longer ears"),
        positive(has,"good natured personality"),!.
dog_is("Cocker Spaniel") :-
        it_is("long-haired dog"),
        positive(has,"height under 22 inches"),
        positive(has,"low-set tail"),
        positive(has,"longer ears"),
        positive(has,"good natured personality"),!.
dog_is("Irish Setter") :-
        it_is("long-haired dog"),
```

```
        positive(has,"height under 30 inches"),
        positive(has,"longer ears"),!.
dog_is("Collie") :-
        it_is("long-haired dog"),
        positive(has,"height under 30 inches"),
        positive(has,"low-set tail"),
        positive(has,"good natured personality"),!.
dog_is("St. Bernard") :-
        it_is("long-haired dog"),
        positive(has,"low-set tail"),
        positive(has,"good natured personality"),
        positive(has,"weight over 100 lb"),!.
it_is("short-haired dog") :-
        positive(has,"short-haired"),!.
it_is("long-haired dog") :-
        positive(has,"long-haired"),!.
```

Додаток 4

```
% Демонстрація роботи експертної системи, що базується на логіці

% Експерт по породам собак
% Демонстрація роботи експертної системи, що базується на логіці,
% складається з бази знань (БЗ), механізму виводу (МВ)
% і системи користувальницького інтерфейсу (СКІ).
% База знань розташовується в оперативній пам`яті
 domains
        CONDITIONS      = BNO *
        HISTORY         = RNO *
        RNO, BNO, FNO   = INTEGER
        CATEGORY        = SYMBOL
 database
        /* Предікати бази даних              */
        rule(RNO, CATEGORY, CATEGORY,CONDITIONS)
        cond(BNO, STRING)
        yes(BNO)
        no(BNO)
        topic(string)
 predicates
        /* Предікати системи користувальницького  інтерфейсу  */
        do_expert_job
        show_menu
        do_consulting
        process(integer)
        info(CATEGORY)
        goes(CATEGORY)
        listopt
```

```
erase
clear
eval_reply(char)
/* Предікати механізму виводу              */
go(HISTORY, CATEGORY)
check(RNO, HISTORY, CONDITIONS)
notes(BNO)
inpo(HISTORY, RNO, BNO, STRING)
do_answer(HISTORY, RNO, STRING, BNO, INTEGER)
goal
do_expert_job.
clauses
/* База знань (БЗ)                         */
topic("dog").
topic("short-haired dog").
topic("long-haired dog").
rule(1, "dog",            "short-haired dog", [1]      ).
rule(2, "dog",            "longt-haired dog", [2]      ).
rule(3, "short-haired dog","English Bulldog ", [3,5,7]   ).
rule(4, "short-haired dog","Beagle",          [3,6,7]   ).
rule(5, "short-haired dog","Great Dane",      [5,6,7,8] ).
rule(6, "short-haired dog","American Foxhound",[4,6,7]   ).
rule(7, "long-haired dog", "Cocker Spaniel",  [3,5,6,7] ).
rule(8, "long-haired dog", "Irish Setter",    [4,6]     ).
rule(9, "long-haired dog", "Collie",          [4,5,7]   ).
rule(9, "long-haired dog", "St. Bernard",     [5,7,8]   ).
cond(1,    "short-haired"            ).
cond(2,    "long-haired"             ).
cond(3,    "height under 22 inches"  ).
cond(4,    "height under 30 inches"  ).
cond(5,    "low-set tail"            ).
```

```
        cond(6,    "longer ears"              ).
        cond(7,    "good natured personality"  ).
        cond(8,    "weight over 100 lb"        ).
        /* Система користувальницького інтерфейсу      */
do_expert_job :-
        makewindow(1,7,7," DOG EXPERT SYSTEM ",0,0,25,80),
        show_menu,
        nl,write(" Press space bar. "),
        readchar(_), exit.
show_menu :-
        write("                                  "),nl,
        write(" * * * * * * * * * * * * * * * * * *  *  "),nl,
        write(" *          DOG EXPERT              *  "),nl,
        write(" *                                  *  "),nl,
        write(" *    1. Consultation               *  "),nl,
        write(" *                                  *  "),nl,
        write(" *                                  *  "),nl,
        write(" *    2. Exit the system            *  "),nl,
        write(" *                                  *  "),nl,
        write(" * * * * * * * * * * * * * * * * * *  *  "),nl,
        write("                                  "),nl,
        write("Please enter your choice: 1 or 2 : "),nl,
        readint(Choice),
        process (Choice).
process(1) :-  do_consulting.
process(2) :-
        removewindow, exit.
do_consulting :-
goes(Mygoal), go([],Mygoal),   !.
        do_consulting :-
        nl, write(" Sorry I can't help yuo."), clear.
```

```
do_consulting.
goes(Mygoal) :-
        clear, clearwindow,
        nl,nl, write("                                  "),nl,
        write("   WELCOME TO THE DOG EXPERT SYSTEM   "),nl,
        write("                                  "),nl,
        write("This is a dog identification system.  "),nl,
        write("To begin the process of choosing a    "),nl,
        write("dog, please type in 'dog'. If you     "),nl,
        write("wish to see the dog types, please     "),nl,
        write("type in a question mark (?).          "),nl,
        write("                                  "),nl,
        readln(Mygoal),  info(Mygoal),!.
info("?") :-
        clearwindow, write("Reply from the KBS."),nl,
        listopt,  nl,write("Please any key. "),
        readchar(_), clearwindow,
        exit.
info(X) :-  X >< "?".
listopt :-
        write("The dog types are : "),nl,nl, topic(Dog),
        write("      ",Dog),nl, fail.
listopt.
inpo(HISTORY,RNO,BNO,TEXT) :-
        write("Question :- ",TEXT," ? "),
        makewindow(2,7,7,"Response",10,54,7,20),
        write("Type 1 for 'yes' ,"),nl,
        write("Type 2 for 'no' : "),nl,
        readint(RESPONSE),  clearwindow,
        shiftwindow(1),
        do_answer(HISTORY,RNO,TEXT,BNO,RESPONSE).
```

```
eval_reply('y') :-
        write(" I hope you have found this helpful !").
eval_reply('n') :-
        write(" I am sorry I can't help you !").
go(_,Mygoal) :-
        not(rule(_,Mygoal,_,_)),!,
        nl,write(" The dog you have indicated is a(n) ",
        Mygoal,"."),nl,
        write("Is a dog you would like to have (y/n) ?"),
        nl,readchar(R),
        eval_reply(R).
        /*          Механізм виводу          */
go(HISTORY, Mygoal) :-
        rule(RNO,Mygoal,NY,COND),
        check(RNO,HISTORY,COND),
        go([RNO|HISTORY],NY).
check(RNO,HISTORY,[BNO|REST]) :-
        yes(BNO),!,
        check(RNO,HISTORY,REST).
check(_,_,[BNO|_]) :- no(BNO),!,fail.
check(RNO,HISTORY,[BNO|REST]) :-
        cond(BNO,NCOND),
        fronttoken(NCOND,"not",COND),
        frontchar(COND,_,COND),
        cond(BNO1,COND),
        notes(BNO1),!,
check(RNO,HISTORY,REST).
check(_,_,[BNO|_]) :-
        cond(BNO,NCOND),
        fronttoken(NCOND,"not",COND),
        frontchar(COND,_,COND),
```

```
        cond(BNO1,COND),
        yes(BNO1),
        !,fail.
check(RNO,HISTORY,[BNO|REST]) :-
        cond(BNO,TEXT),
        inpo(HISTORY,RNO,BNO,TEXT),
        check(RNO,HISTORY,REST).
check(_,_,[]).
notes(BNO) :- no(BNO),!.
notes(BNO) :- not(yes(BNO)),!.
do_answer(_,_,_,_,0) :- exit.
do_answer(_,_,_,BNO,1) :-
        assert(yes(BNO)),
        shiftwindow(1),
        write(yes),nl.
do_answer(_,_,_,BNO,2) :-
        assert(no(BNO)), write(no),nl, fail.
erase :- retract(_),fail.
erase.
clear :- retract(yes(_)),retract(no(_)),fail,!.
clear.
```

Додаток 5

```
domains
      mesto=a(integer,integer,symbol)
      spisok= mesto*
predicates
      a(integer,integer,symbol)
      путь(mesto,spisok,spisok)
      member(mesto,spisok)
      можно_идти(mesto,spisok)
clauses
                  a(1,1,стена).a(1,2,стена).a(1,3,стена).a(1,4,стена).
                  a(2,1,пусто).a(2,2,пусто).a(2,3,пусто).a(2,4,стена).
a(3,1,выход).a(3,1,пусто).a(3,2,стена).a(3,3,пусто).a(3,4,стена).
                  a(4,1,стена).a(4,2,стена).a(4,3,пусто).a(4,4,пусто).
                  a(5,1,стіна).a(5,2,пусто).a(5,3,пусто).a(5,4,стіна).
% a(I,J) представляє позицію в і рядку j стовбці
%  знайшли шлях?
 путь(a(I,J,B),[a(I,J,B)],Были):-
       a(I,J,выход),nl,write("Выход",I,J).
% Намагаємося йти на північ
 путь(a(I,J,B),[a(I,J,B)|P],Были) :-
        K= I-1,
        можно_идти(a(K,J,B),Были),
        путь(a(K,J,B),P,[a(K,J,B)|Были]).
% Намагаємося йти на південь
 путь(a(I,J,B),[a(I,J,B)|P],Были) :-
        K= I+1,
        можно_идти(a(K,J,B),Были),
        путь(a(K,J,B),P,[a(K,J,B)|Были]).
% Намагаємося йти на захід
```

```
путь(a(I,J,B),[a(I,J,B)|P],Были) :-
        L= J-1,
         можно_идти(a(I,L,B),Были),
        путь(a(I,L,B),P,[a(I,L,B)|Были]).
% Намагаємося йти на схід
путь(a(I,J,B),[a(I,J,B)|P],Были) :-
        L= J+1,
        можно_идти(a(I,L,B),Были),
        путь(a(I,L,B),P,[a(I,L,B)|Были]).
%  в позицію і j можно потрапити якщо це не стіна і ми не були в ній раніше
можемо_йти(a(I,J,B),Були):-
        a(I,J,пусто),
% перевірити були чи ні?
       not (member(a(I,J,_),Были)),
       nl.
member(R,[R|T]).
member(R,[H|T]):-member(R,T).

goal

    путь(a(4,2,пусто),P,[a(4,2,пусто)]), write(P).
```

Література

1. Aikins J.S. Prototypical knowledge for expert systems//Artificial Intelligence.-1983.v.20. p.163-210.
2. Baldwin J. E., eds. (1996). Fuzzy Logic. New York: Wiley.
3. Clements B.R. and Preto F. Evaluating Commercial Real Time Expert System Software for Use in the Process Industries. – C&I. – 1993. – p.107-114.
4. Feigenbaum E.A. The art of artificial intelligence: Themes and case studio of knowledge engineering// The fifth International Joint Conference on Artificial Intelligence/ -Boston: MIT, 1977.-p.1014-1029.
5. Hall C. The Intelligent Software Development Tools Market//Part I.. Intelligent Software Strategies. 1996. February. V.12.- № 2. – p.1 – 12.
6. Hall C. The Intelligent Software Development Tools Market//Part II. Intelligent Software Strategies. 1996. V.12.- № 3. – p.1 – 16.
7. Harmon P. The Market for intlligent Software Products Intelligent Software Strategies, 1992. V.8. 2. – P. 5-12. Intelligent Software Development Tools Market//Part I.. Intelligent Software Strategies. 1995. V.11.- № 2. – p.1 – 13.
8. Harmon P. The Size of Commercial Al Market in the US//Part II. Intelligent Software Strategies. 1994. V.10.- №1. – p.1 – 6.
9. Hopfield J.J. Neural Network and Physical Systems with Emergent Collective Computation Abilities. //Proc. Nat. Acad. Science USA, 1982-V 79, Pp.2445-2558.
10. Intelligent Software Stretegies. №2. – 1996.
11. Jamshidi M., Tilti A., Zadeh L. And Boverie S., eds. (1997). Applications of Fuzzy Logic: Towards High Machine Intelligence Quotient Systems. Englewood Cliffs, NJ: Prentice Hall.
12. Kowalski, R. Algorithm=logic+control. Communications of the ACM, vol. 22, no.7, pp. 424-436, July 1979.(Prolog)

13. Kowalski, R. Logic for problem solving. New York: American Elsevier, 1979.(Prolog)

14. Minsky M. A framework for representation knowlege//Psychology computer vision. – New York: McGraw-Hill, 1975

15. Moore D. and other., Questions and Answers about G2//Copyright.- 1993. Gensym Corporation. p. 26-28.

16. Robinson J. A. (1979) Logic: Form and Function. Edinburgh: Edinburgh University Press.

17. Адаменко А.Н., Кучуков А.М. Логическое программирование и Visual Prolog. – СПб.: БХВ-Петербург, 2003.-992 с.: ил.

18. Айзерман М.А., Браверман Э.М., Розоноэр Л.И. Метод потенциальных функций в теории обучения машин. – М.: Наука, 1970. –383с.

19. Барцев С.И., Охонин В.А. Адаптивные сети обработки информации Красноярск: Ин-т физики СЩ АН СССР, 1986, Препринт 59Б.-20с.

20. Бонгард М.М. Проблема узнавания. М.:Наука.- 320с.

21. Бондарев В.Н. Искусственный интеллект: Учеб.пособие для вузов/ В.Н.Бондарев, Ф.Г. Аде - Севастополь: Изд-во СевНТУ, 2002.-615с.: ил.

22. Братко И. Программирование на языке Пролог для искусственного интеллекта: Пер. с англ. – М.: Мир, 1990. – 560 с., ил.

23. В. О. Сафонов. Экспертные системы- интеллектуальные помощники специалистов. С.-Пб: Санкт-Петербургская организация общества “Знания” России, 1992.

24. Вагин В.М. Дедукция и обобщение в системах принятия решений. - М.: Наука, 1988. - 384с.

25. Гейн К., Сарсонт Т. Структурный системный анализ: средства и методы. В 2-х ч. Ч.1/Пер. с англ. Под ред. А.В. Козлинского. – М.:Эйтекс, 1993. – 188 с.

26. Голицын Г.А., Петров В.М. Информация- поведение– творчество. М.: Наука, 1991.- 224с.

27. Голицын Г.А., Фоминых И.Б. Интеграция нейросетевой технологии с експертными системами // Труды 5 Национальной конференции по ИИ – Казань, 1996.

28. Горбань А. Нейроинформатика и ее приложения/Открытые системы, 4, 1998.

29. Горбань А. Современные направления развития нейрокомпьютерных технологий в России /Открытие системы, 4, 1997, стр. 25-28.

30. Грей П. Логика, алгебра и базы данных/Пер. с англ. Под ред. Г.В. Орловского, А.О. Слисенко.- М.: Машиностроение, 1989,- 368 с.: ил.

31. Д. Н. Марселлус. Программирование экспертных систем на Турбо Прологе.- М.: Финансы и статистика, 1994.

32. Д. Уотермен. Руководство по экспертным системам. М.: Мир, 1980.

33. Д. Элти, М. Кумбс. Экспертные системы: концепции и примеры. М.: Финансы и статистика, 1987.-191 с.

34. Дж. Доорс, А. Р. Рейблейн, С. Вадера Пролог – язык программирования будущего; Пер. с англ.; -М.: Финансы и статистика 1990. 144 с.:ил.

35. Джексон П. Введение в экспертные системы: Пер. с англ.: Учебное пособие/- М.; Издательский дом Вильямс, 2001.- 624 с.

36. Джорж Ф.Люгер Искусственный интеллект: стратегии и методы решения сложных проблем, 4-е издание.: Пер. с англ.- М.: Издательский дом “Вильямс”, 2003.-864 с.

37. Дуда Р., Харт П. Распознавание образов и анализ сцен. – М. : Мир, 1976. –512с.

38. Ивахненко А.Г. Персептроны. - Киев: Наукова думка, 1974.

39. Ин Ц., Соломон Д. Использование Турбо-Пролога: Пер. с англ.-М.:Мир, 1990.- 300 с.

40. Искусственный интеллект. - В 3-х кн. Кн. 1. Системы общения и экспертные системы: Справочник/Под ред. Э.В. Попова.-М.: Радио и связь, 1990.-464 с.: ил.

41. Искусственный интеллект: В 3 кн. Кн.2. Модели и методы: Справочник/Под ред. Д.А. Поспелова - М.: Радио и связь, 1990. – 304 с.

42. К. Нейлор. Как построить свою экспертную систему М.: Энергоатомиздат, 1991.- 286 с.

43. К. Таунсенд, Д. Фохт. Проектирование и программная реализация экспертных систем на персональных ЭВМ.- М.: Финансы и статистика, 1990.

44. Малпас Дж. Реляционный язык пролог и его применение: Пер. с англ./Под редакцией В.Н. Соболева. – М.: Наука. Гл. ред. физ-мат.лит., 1990. – 464 с.

45. Маслов С.Ю. Обратный метод установления выводимости в классическом исчислении предикатов//Доклады Академии наук СССР.Т 159.–1964. №1.С.17-20.

46. Мендельсон Э. Введение в математическую логику. – М.: Наука, 1971.-320с.

47. Минский М. На пути к созданию искусственного разума//Вычислительные машины и мышление. – М.: Мир, 1967. – 552с.

48. Минский М. Структура для представления знания. –Сб. Психология машинного зрения. Под ред. П. Уинстона. М.: Мир, 1978.- с.249-338.

49. Минский М.Л., Пейперт С. Персептероны. – М.: Мир, 1971.

50. Нечеткие множества в моделях управления и искусственного интеллекта. /Под редакцией Д.А. Поспелова. - М.: Наука, 1986.- 312 с.

51. Нильсон Н. Искусственный интеллект. Методы поиска решений. – М.: Мир, 1973.-265 с.

52. Нильсон Н. Принципы искусственного интеллекта. – М.: Радио и связь, 1985.- 376 с.

53. Обработка нечеткой информации в системах принятия решения/А.Н. Борисов, А.В. Алексеев, Г.В. Меркурьев.- М.: Радио и связь, 1989. – 304с.

54. Осуга С. И др. Приобретение знаний/Пер. с япон.под рук.С.Осуги. - М.: Мир, 1990,-304 с.: ил.

55. Осуга С. Обработка знаний/Пер. с япон. – М.: Мир, 1989.- 293с., ил.

Осуга С. Обработка знаний: Пер. с япон. - М.: Мир, 1989. – 293 с.

56. Петров Э.И. Система Rethink. Применение. Аспекты//Материалы семинара “Динамические интеллектуальные системы в управлении”. - М: ЦРДЗ, 1996. – с.58-64.

57. Попов Э.В. Фоминых И.Б. Кисель Е.Б. Статические и динамические экспертные системы.- М: ЦРДЗ, 1995. – 126 с.

58. Попов Э.В. Экспертные системы реального времени//Материалы семинара “Экспертные системы реального времени ”- М: ЦРДЗ,1995.– с.5-22.

59. Попов Э.В. Экспертные системы. Решение неформализованных задач в диалоге с ЭВМ. – М.: Наука, 1987.

60. Попов Э.В., Фридман Г.Р. Алгоритмические основы интеллектуальных роботов и искусственного интеллекта. – М.: Мир, 1976. – 455с.

61. Поспелов Г.С. Искусственный интеллект. Новая информационная технология//Вестник АН СССР. – 1983. - №6. – С. 31-42.

62. Представление и использование знаний: Пер. с япон. / Под ред. Х. Уэно, М. Исидзука. - М.: Мир, 1989. – 220 с.

63. Рот М. Интеллектуальный автомат: компьютер в качестве эксперта/ Пер. с нем. А.П. Свиридова. - М.: Энергоатомиздат, 1991

64. Статические и динамические экспертные системы: Учебное пособие/ Э.В. Попов и др. М.: Финансы и статистика, 1996. -320с.: ил.

65. Т.А. Гаврилова, В.Ф. Хорошевский. Базы знаний интеллектуальных систем.- СПб: Притер, 2000. – 384 с.: ил.

66. Таусенд К., Фохт Д. Проектирование и программная реализация экспертных систем на персональных ЭВМ /Пер с англ. В.А. Кондратенко. - М.: Финансы и статистика, 1990

67. Тейз А., Грибомон П., Луи Ж. И др. Логический подход к искусственному интеллекту: от классической логики к логическому программированию/Пер с франц.-М.: Мир, 1990.-43 с.: ил.

68. Тейлор К. Как построить свою экспертную систему/ Пер. с англ. Н.Н. Слепова. - М.: Энергоатомиздат, 1991

69. Уинстон П. Искусственный интеллект. М.1980.

70. Уотермен Д. Руководство по экспертным системам: пер. с англ./ Под ред. В.Л.Стефанюка -М.: Мир, 1989

71. Хант Э. Искусственный интеллект. -М.: Мир,1978.

Навчальне видання

КРАВЕЦЬ Валерій Олексійович

ХАВІНА Інна Петрівна

КОЛИБІН Юрій Миколайович

НІКІТІНА Людмила Олексіївна

ЗИКОВ Ігор Семенович

ФІЛОНЕНКО Алевтина Михайлівна

ХАВІН Валерій Львович

ВСТУП ДО ЕКСПЕРТНИХ СИСТЕМ

Роботу до видання рекомендував В. Д. Дмитрієнко

Навчальний посібник

Редактор О.І. Шпильова

План 2004 р., п. 70.

Підписано до друку _____. Формат 60х84 1.16. Друк – ризографія.

Папір офсетний. Гарнітура Times New Roman. Умов. друк. арк. 12,6. Обл.-вид.арк..13,9.

Тираж 300 прим. Зам. № _____. Ціна договірна.

Видавничий центр НТУ ”ХПІ”, 61002, Харків, вул. Фрунзе, 21.

Свідоцтво про державну реєстрацію ДК № 116 від 10.07.2000 р.

Друкарня НТУ “ХПІ”, 61002, вул. Фрунзе, 21.